PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-243089

(43)Date of publication of application : 07.09.2001

(51)Int.Cl. G06F 11/28

(21)Application number : 2000-049146

(22)Date of filing : 25.02.2000

(71)Applicant : MITSUBISHI ELECTRIC CORP

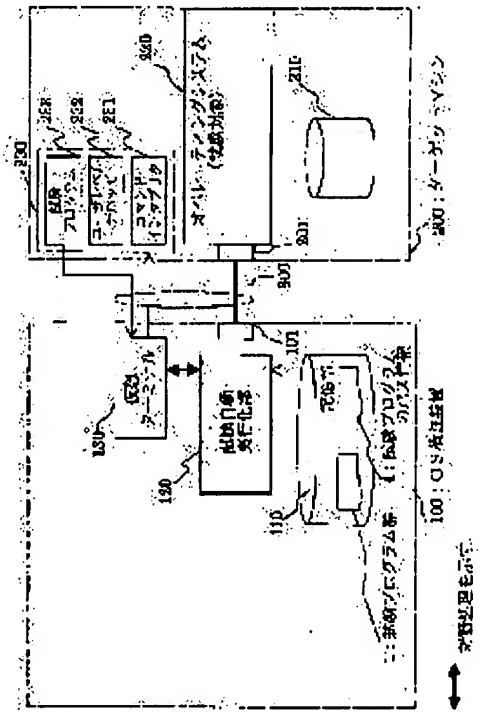
(72)Inventor : BABA YOSHIYUKI

(54) DEVICE AND METHOD FOR VERIFYING SOFTWARE

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a verifying device for easily searching the cause of the abnormality of software to be tested.

SOLUTION: A software verifying device is provided with a test program storing part 110 for storing a test program for testing software by using the API of software to be tested and a test automatic executing part 120 for starting a debugger by designating the test program stored in the test program storing part 110 as a program to be operated by using the debugger, and for setting a break point at the part of the designated test program for allowing the test program to grasp an error by the API of the software by using the started debugger, and for executing the set test program by using the started debugger.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or

application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention is a technique about the equipment and the approach of examining OS (Operation System) of a computer. In the verification equipment which checks the contents of service of OS using the application programming interface which OS offers especially, the trial automatic activation approach of making easy cause pursuit at the time of test failure is offered.
[0002]
[Description of the Prior Art] There are the test approach and program test equipment which are indicated by JP,8-305609,A in the conventional approach for automating the test of a program. An example of the structure of a system is conventionally shown in drawing 31 . Drawing 31 shows the relation between program test equipment 10, the target system 20 as a means to perform the program for a test, and a tester 30, for example, an operator. Program test equipment is equipped with test script management tool 10a, test automation control means 10b, test script activation means 10c, and 10d of test yes-no decision means and program failure specification means 10e. Furthermore, this program test equipment 10 is equipped with 10f of test script input means, 10g of test-result output means and 10h of mode change / activation means, and debug command input means 10i as an interface means between testers 30. These interface means 10f-10i are connected with test automation control means 10b, respectively.
[0003] In the conventional technique, program test actuation has become as follows. While preparing the test script which includes at least the purport to which the activation result in the predetermined step in the suitable program for a test for the judgment of whether a failure is in the program for a test is made to output, and the test entry over this activation result and performing the program for a test, it judges [whether a failure is to the program for a test, and] by processing of said test script in a predetermined step.
[0004] Furthermore, there is failure specification actuation in the conventional technique. This is performed as follows. Since the tester included the failure, he displays on the display of program test equipment 10 the test script for a failure existence judging to the program made into the rejection in program test actuation using 10f of test script input means. Next, the suitable test script for failure specification for failure part specification of a program including this failure is registered into the file of the storage section of test script management tool 10a using 10f of test script input means.
[0005] When judged with this test script for failure specification to register being with obstacles by the static test mode (citation abbreviation from the above-mentioned official report), it sees from the obtained test result, and the test script for a failure existence

judging is changed and created so that it may be suitable for specification of the failure concerned. For example, the break point and the data name which should be dumped are changed. Then, a part or all of a test program is rerun. As a result, it was presupposed by processing of the test script for failure specification that a failure part can be pinpointed.
[0006]
[Problem(s) to be Solved by the Invention] However, in the test method which registers and reruns the script which pinpoints a failure part anew like the conventional example once detecting a failure, the technical problem that cause pursuit was difficult occurred to the failure that repeatability is low. Made in order that this invention might solve the above technical problems, in the equipment which verifies a test objective program automatically using a test program, the 1st purpose makes it possible to set up a break point on a test program, and stops test actuation.
[0007] The 2nd purpose is in the situation for the location where a test program detects a failure for the break point set up for the 1st purpose, and acquires the information for analyzing the cause of a failure at the time of failure detection about a failure with difficult reappearance by acquiring the internal state of a test program after a test program halt.
[0008] The 3rd purpose is in the situation for the location where the test program detected the failure for the break point set up for the 1st purpose, and acquires the information for analyzing the cause of a failure at the time of failure detection about a failure with difficult reappearance by acquiring the internal state of a test objective program after a test program halt.
[0009] The 4th purpose is the resource with which a test program is in the situation for the location which detected the failure, and a test program uses the break point set up for the 1st purpose, and acquires the information about the execution environment of a test program which influences the activation result of a test program because processing of an operating system (it is hereafter described as OS) changes according to the condition.
[0010] The 5th purpose judges that the test program caused the hang-up, when time-out judging time amount is defined and a test program is not completed in this time amount, after it carries out automatic collection of the data for analyzing the factor which caused the hang-up, terminates a test program and makes consecutive processing of test actuation possible.
[0011] The 6th purpose makes it possible to set up time-out judging time amount efficiently in the 5th purpose.
[0012] The 7th purpose performs data collection for analyzing the factor which prevented experimental automatic activation stopping by the abnormal stop of a test program, and caused the abnormal stop.
[0013] The 8th purpose makes consecutive processing of test actuation possible, after OS of a test objective resets in hardware the target machine with which OS of a test objective operates in the situation that a trial fails in a serious failure by the lifting and having carried out the system down and a target machine reboots by activation of a test program.
[0014]
[Means for Solving the Problem] The software verification equipment concerning this invention is software verification equipment which can start DEBAKKA which examines software. The test program storage section which memorizes the test program which examines the above-mentioned software using the application programming interface

(henceforth "API") of the software made into a test objective, Specify the test program memorized by the above-mentioned test program storage section as a program operated using DEBAKKA, and above-mentioned DEBAKKA is started. A break point is set to the part of a test program where the specified test program grasps an error by API of the above-mentioned software using started DEBAKKA. It is characterized by having the trial automatic activation-ized section performed using DEBAKKA which had the set-up test program started.

[0015] The above-mentioned test-program storage section memorizes the definition about activation of a test program including the error-detection point of a test program further including the error-detection point which grasps that actuation of software is unusual, and it carries out that the above-mentioned trial automatic activation-ized section sets up a break point to a test program based on the error-detection point of the test program memorized by the above-mentioned test-program storage section as the description by the activation result as which the above-mentioned test program was defined by the API of the above-mentioned software.

[0016] The above-mentioned software verification equipment is characterized by having the fault information collection section which collects the fault information at the time of a test program stopping, when a test program stops further by the break point by which a setup was carried out [above-mentioned].

[0017] The above-mentioned fault information collection section is characterized by collecting the values of the variable of a test program based on the variable information which extracted and extracted the above-mentioned variable information using DEBAKKA from the above-mentioned test program including the variable information as information that the variable which uses the above-mentioned test program while a test program performs is specified.

[0018] The above-mentioned fault information collection section is characterized by collecting the values of the variable of a test program based on the test objective variable information that the above-mentioned test objective variable information was extracted and extracted, using DEBAKKA from the above-mentioned software including the test objective variable information as information that the variable which uses the above-mentioned software while software performs is specified.

[0019] The above-mentioned fault information collection section changes the execution environment of the above-mentioned software, and is characterized by acquiring the activation result of the above-mentioned software performed by performing the above-mentioned software on the changed execution environment.

[0020] The above-mentioned trial automatic activation-ized section stops a test program using above-mentioned DEBAKKA, when a test program is not completed after predetermined time amount progress, and the above-mentioned fault information collection section is characterized by collecting fault information while it acquires the halt location of the stopped test program.

[0021] The above-mentioned trial automatic operation-ized section is characterized by changing the above-mentioned predetermined time amount based on the activation result of a test program.

[0022] The test program which detected that the test program carried out the abnormal stop of the above-mentioned trial automatic activation-ized section, and was detected using DEBAKKA is forced to terminate, and the above-mentioned fault information

collection section is characterized by collecting fault information while it acquires the forced-termination location of the test program forced to terminate.
[0023] The above-mentioned trial automatic activation-ized section sends out the signal which directs reset of the computer by which software operates after a test program carries out an abnormal stop, and after the above-mentioned computer reboots with the sent-out signal, it is characterized by performing a test program.
[0024] The above-mentioned software is characterized by including either of an operating system or an application program.
[0025] The software verification approach concerning this invention is the software verification approach which can start DEBAKKA which examines software. The test program storage process of memorizing the test program which examines the above-mentioned software using the application programming interface (henceforth "API") of the software made into a test objective, Specify the test program memorized at the above-mentioned test program storage process as a program operated using DEBAKKA, and above-mentioned DEBAKKA is started. A break point is set to the part of a test program where the specified test program grasps an error by API of the above-mentioned software using started DEBAKKA. It is characterized by having a trial automatic activation chemically-modified [which is performed using DEBAKKA which had the set-up test program started] degree.
[0026]
[Embodiment of the Invention] With the gestalt 1 of gestalt 1. implementation of operation, in the situation of verifying OS by activation of a test program, a test program stops a test program immediately after failure detection, and offers the device carried out to having made the condition at the time of failure detection hold freely. Then, the gestalt of this operation explains as an example OS verification equipment which verifies OS as software verification equipment. Moreover, in the gestalt 8 of operation, OS verification equipment is similarly explained as an example from the gestalt 2 of operation. Therefore, the test objective program used as a test objective serves as OS.
[0027] Drawing 1 expresses the block diagram of an example of a system which realizes the gestalt 1 of operation. 100 expresses OS verification equipment as an example of software verification equipment. OS verification equipment (100) supports the trial of OS performed on a target machine in this invention. OS verification equipment (100) is constituted from the gestalt of this operation by the interface (101) with the channel linked to the storage section (test program storage section) (110), the trial automatic activation-ized section (120), a virtual terminal (130), and a target machine.
[0028] 110 is the storage section (secondary storage) as the test program storage section. Here, it is the file (it is also only called "the pass information on a test program") (4) showing the pass information on a test program that the positional information on the secondary storage of a test program (pass) was expressed as the test program group (1) carried out on a target machine. For details, it is shown in drawing 2 and drawing 3 . 120 is the trial automatic activation-ized section in OS verification equipment, and performs actuation shown in drawing 5 . 130 is a virtual terminal and connects with the group (230) of the test program as a session who operates the test program which operates on OS of a test objective by communication facility. A virtual terminal (130) notifies the group (230) of a test program of the directions from the trial automatic activation-ized section (120).

[0029] 200 expresses the target machine. Besides, OS (220) of a test objective operates. 201 is an interface with the channel linked to OS verification equipment. 210 is the secondary storage on a target machine. 220 is OS (OS program) of a test objective. 230 is a group of a test program who operates on OS (220) of a test objective. 231 is the command interpreter which operates on OS of a test objective, and is a reader rank first started within the group (230) of a test program. As for a command interpreter, after test program termination resides permanently on a target machine, as long as a virtual terminal (130) and connection are maintained.
[0030] 232 is a user level debugger and is started on a command interpreter (231). 233 is a test program and is started through a user level debugger (232). 300 is a channel and has connected the target machine (200) with OS verification equipment (100). As a medium, RS232C and Ethernet are used, for example. Using this channel, a test program is transmitted to a target machine from OS verification equipment, or it is possible to turn to OS verification equipment the standard input/output of a program which operates on a target machine.
[0031] Drawing 2 expresses signs that a test program is managed by the folder according to trial item. The folder (1a) for every trial item has the layered structure, and the src folder (2) which stores the source file for generating a test program, and the bin (3) folder which stores a file at the time of activation of a test program exist as a low-ranking folder hierarchical [this folder]. The source file (21) of a test program and the file (22) which described the procedure which generates a source file blank test program are in a src folder (2). Into a bin folder (3), the definition (5) about the load module (31) of the test program generated from the source file (21) of said test program and activation of a test program is stored.
[0032] OS verification equipment can grasp the definition (5) about activation of the test program to the load module (31) of a test program by defining the definition (5) about activation of a test program as the load module name of a test program by the identifier of ".run."
[0033] Drawing 3 shows the file (4) which describes the pass information on a test program, and its format. In the file which describes the pass information on a test program, the pass (41) of a test program is described by one line at one rate. With the gestalt of this operation, since a user level debugger is minded at the time of test program activation, a source file is needed at the time of activation of a test program. The test program is managed by the folder which has a layered structure as shown in drawing 2 , during the pass (41) of a test program, the file name of the load module of a test program and a that front serve as a bin folder, and a that front serves as [the last] a trial subject name. An example "/kernel/syscall/read/" of a trial subject name is shown in drawing 3 . Thereby, it enables the trial automatic activation-ized section (120) to transmit the contents of the required folder (1a) for every trial item to a target machine according to the gestalt of the user level debugger in a target machine of operation at the time of test program activation.
[0034] Or the contents of the bin folder under this (3) may be transmitted. It is the cross development host of a target machine (200), and the body of a user level debugger on OS verification equipment (100) operates on OS verification equipment (100), and, as for the processing which transmits only the contents of the bin folder (3), OS verification equipment (100) corresponds to the test atmosphere to which the part and test program of

a user level debugger operate on a target machine (200). OS verification equipment processes every one entry of the test program in the file (4) which describes the pass information on a test program. Thereby, OS verification equipment grasps the test program under operation on a target machine.

[0035] Drawing 4 shows the definition (5) about test program activation, and its format. The definition (5) about test program activation exists at one rate to a test program in the folder (1a) for every trial item. It is the mechanism in which the trial automatic activation-ized section (120) can grasp correspondence, by the extension (.run) to a test program name in principle. There is the following information which is needed in case a test program is performed on this OS verification system in the definition (5) about test program activation.

[0036] 51 is the argument information at the time of test program activation. The gestalt of this operation is describing to the head line. 52 is the information about the break point set up at the time of test program activation. The gestalt of this operation has indicated using one line to one break point after the 2nd line. In case the information (52) about a break point displays in a list on a debugger the file which constitutes a test program, it consists of the line number (54) in the function name (53) used as a keyword, and the file which constitutes a test program. In addition, a test implementation person chooses the setting location of the break point in a test program in principle as a location where the test program detected the error.

[0037] Drawing 5 shows the flow of processing of the trial automatic activation-ized section in OS verification equipment which realizes the gestalt 1 of operation. Below, processing at each step is explained.

[0038] At step 900, a virtual terminal (130) is prepared on OS verification equipment (100), and using the communication facility of a virtual terminal, starting of a command interpreter (231) is required on a target machine (200), and it connects with the generated command interpreter (231). Through the actuation on this virtual terminal (130), on a target machine (200), the various commands of a user level debugger (232) and a test program (233) are started, and OS (220) of a test objective is examined.

[0039] At step 1000, the pass information on a test program (4) is read. For example, in the trial automatic activation-ized section (120), it is possible to treat by passing the file name which describes the pass information on a test program (4) as an argument at the time of starting, or setting it as the pass of immobilization. The trial automatic activation-ized section (120) is one loop-formation processing from step 1000 to step 1700, and treats one test program at a time. For this reason, at step 1000, the entry of a test program is read from a head in order in the pass information on a test program (4).

[0040] At step 1100, processing is changed by whether there is any trial which carries out based on the pass information read by processing at step 1000, and is carried out, or there is nothing. If description of the trial to perform does not exist, it will say having reached to the last line in the pass information (4) file of a test program. When description of the trial which moves to step 1200 and is performed when description of the trial to perform exists does not exist, processing of the trial automatic activation-ized section is ended.

[0041] At step 1200, it realizes using the means (henceforth a "automation means") which automated that an operator produced effectiveness equivalent to carrying out a key type to the virtual terminal (130) prepared at step 900. Since, as for the character string across which it goes by the processing of a key type performed to a virtual terminal

(130), the command interpreter on a virtual terminal (130) and a target machine (200) is connected, the contents are interpreted by the command interpreter. Although the purpose here is starting a test program (233) through a user level debugger (232) on a target machine (200), as this is the following, it is performed.

[0042] First, a test set required in order to perform a test program on a target machine is transmitted to a target machine. For example, a command interpreter is made to interpret activation of a remote copy command, and the whole folder (1) for every trial item is transmitted to the secondary storage (210) on a target machine. The pass of the folder which should be transmitted which is needed at the time of remote copy command execution can be obtained from the pass information on a test program (4) as pass except a test program name and below a bin folder. Specifically within OS verification equipment, the character string pathname .(line feed)" of the folder which should be carried out a "rcp -r OS verification device name:transfer is sent on a virtual terminal (130). Character string **** which consists of a test program name after the pass of a user level debugger, and it to up to a virtual terminal (130) after completing a transfer of the test program to a target machine top. This character string is interpreted by the command interpreter on a target machine, and a test program is started through a user level debugger on a target machine. At this time, the test program has not carried out activation initiation yet.

[0043] At step 1300, it is the processing which reads the definition (5) file about activation of a test program. By this processing, the information (52) about the argument at the time of test program activation (51) and the break point set up into a test program is acquired. With the gestalt of this operation, as shown in drawing 2 , the file of the definition (5) about activation of a test program is defined as what attached the extension ".run" to the pass of the test program under present activation, and exists in the folder in which a test program exists.

[0044] At step 1400, a break point is set as a test program (233) using the automation means indicated to step 1200 on the user level debugger started on the target machine (200) by processing of step 1200. Here, a break point is set as the line number (54) in the file which is made to read the source file which constitutes a test program using 53 among the information (52) about the break point set up into the test program read to the user level debugger at step 1300, next constitutes a test program. When the multi-statement of the break point needs to be carried out, this actuation on a user level debugger is repeated and performed.

[0045] You make it accompanied by the argument at the time of the test program activation which obtained the test program which started with the user level debugger at step 1300 using the automation means indicated to step 1200, and it is made to start at step 1500. "runarg1 arg2 arg3 (return)" is sent in the example of the character string which the user level debugger on a target machine interprets [character string], and makes a test program specifically start with a desired argument to a virtual terminal (130), and drawing 4 .

[0046] At step 1600, it is the processing which supervises termination of a test program (233) by automatic dialogue actuation (screen capture) on a virtual terminal (130). This processing carries out the capture of the screen of a virtual terminal (130) to every fixed time amount (for example, 1 second), and obtains alphabetic data to the string array according to the size of a virtual terminal. For example, processing which incorporates

the alphabetic character currently outputted to the array of 80 figure x24 line on the virtual terminal (130) is performed. In the situation that the standard output of a test program and a user level debugger is displayed on a virtual terminal (130), and the contents of a display are specifically scrolled upward a tester With the prompt of the user level debugger displayed on the lowest line on a screen (the 24th line), and having been outputted before that, the information which shows the condition of the test program by the user level debugger which should be displayed on one line (the 23rd line) is acquired as character-string data, and is supervised. An experimental activation situation is grasped by performing string-comparison processing for the character-string data for two above-mentioned lines. Since the prompt of a debugger incidentally is not displayed before the test program is completed, the character-string data equivalent to the 24th line on the screen which carried out the screen capture are not the character strings showing the prompt of a debugger.

[0047] At step 1700, based on processing at step 1600, when a test program (233) stops by the break point, it ends. When a test program does not stop by the break point, it returns to step 1000. The character-string data which are equivalent to the 24th line on a screen in the data which carried out the screen capture of the virtual terminal (130) by processing of step 1600 as an example which shows the condition at the time of having stopped by the break point are the same as the prompt of a user level debugger, the character-string data equivalent to the 23rd more line show the break location, and it is shown that the character-string data equivalent to the 22nd more line stopped by the break point.

[0048] If a break point is not detected during activation of a test program, based on the pass information on a test program (4), automatic activation of the test program is carried out one by one by processing of the above trial automatic activation-ized section. On the other hand, when a break point is detected, a test program is having stopped with as by the break point set up by processing of step 1400. Thus, activation of a test program is stopped in the part of a request of a test program.

[0049] Thus, the operating system of the calculating machine made into a test objective In the software verification equipment verified with the test program which checks the contents of service of OS using the application program MINGU interface which OS offers By carrying out activation initiation, after setting a break point as the part in which starts said test program through the user level debugger realized on OS, and a test program makes an error A test program is stopped after failure detection and it is characterized by carrying out to having made freely the condition of having detected the failure hold.

[0050] With the gestalt 2 of gestalt 2. implementation of operation, it carries out like the gestalt 1 of the above-mentioned implementation, and when it stops by the break point to which the test program was set, the case where the collection processor of the fault information which acquires the value (argument information) of the variable which the test program uses is realized is explained. With the gestalt of this operation, although the test program has acquired the internal state of the test objective program at the time of failure detection, this makes avoidable the problem on which a test objective program is judged to be a failure by the grasp of procedure from which a test objective program starts a failure, and the error of a test program. OS verification equipment is explained as an example like [the gestalt of this operation] the gestalt 1 of operation.

[0051] Drawing 6 is the block diagram of an example of a system which realizes the gestalt 2 of operation. 140 is the fault information collection section in OS verification equipment, and performs actuation shown in drawing 9 . 111 is fault information. The fault information in the gestalt of this operation is data of the test program under halt on the target machine collected by the fault information collection section (140). Fault information is related with the identifier of the test program under halt on a target machine, and is recorded on the secondary storage of OS verification equipment. In drawing 6 , although fault information (111) is expressed as storage different from the storage section (110), it may be the same storage.
[0052] Drawing 7 shows the file (6) and format of the test program symbol information that the symbol information inside a test program is expressed. Symbol information shows the address with which the head (it is called a symbol) of the data used inside the program and a function is assigned, when a program is developed on memory. Usually, after compiling the source file of a program, the address assigned is decided, when link processing is carried out. Symbol information is included in the load module. With the gestalt of this operation, the fault information collection section (140) acquires symbol information if needed. About this, it mentions later (drawing 9 , step 1722). For example, in the criterion UNIX ("UNIX" is a trademark), the file (6) showing the symbol information inside a test program is in the condition that a symbol table is contained in the load module of a test program, and is obtained by specifying and performing the pass of the load module of a test program to the argument of a standard nm command.
[0053] 61 expresses the address of a symbol. 62 expresses the identifier of a symbol. The identifier of a symbol shows whether a symbol is data or it is the head of a function. The symbol which is data is distinguished for information here. 63 expresses a symbol name. When the class of symbol is data, it is the identifier of a variable data. The variable data obtained from this symbol information serves as an object which collects the values in that time as fault information, when a test program stops by the break point.
[0054] Drawing 8 shows drawing showing an example of the flow of processing of the trial automatic activation-ized section (120) which realizes the gestalt 2 of operation. To drawing 5 , this processing changed processing of step 1700, and was used as step 1710, and processing of step 1720 is added. Processing at the step these-changed into below is explained.
[0055] At step 1710, processing is changed by whether the test program (233) stopped by the break point set up by processing of step 1400. When having stopped by the break point, it moves to processing of step 1720. When it ends without stopping by the break point, it returns to processing of step 1000. The character-string data which are equivalent to the 24th line on a screen in the data which carried out the screen capture of the virtual terminal (130) by processing of step 1600 as an example which shows the condition at the time of having stopped by the break point are the same as the prompt of a user level debugger, the character-string data equivalent to the 23rd more line show the break location, and it is shown that the character-string data equivalent to the 22nd more line stopped by the break point.
[0056] The fault information collection section (140) is processed at step 1720. The flow of drawing 9 shows the detail of processing of step 1720.
[0057] Next, drawing 9 is explained. It is the processing which gains the halt line number of a test program (233), and leaves fault information (111) by automatic dialogue

actuation (screen capture) on a virtual terminal (130) at step 1721. When having stopped by the break point set up on the user level debugger, in the data which carried out the screen capture, the purport and halt positional information (the source file and halt line number of a test program) which stopped with the halt line number to the prompt of a user level debugger and the 23rd line, and were stopped by the break point to the 22nd line at the 24th line on a virtual terminal (139) are shown.

[0058] At this step, although it leaves fault information (111) to a file, the information acquired by the 22 or 23rd line in the obtained data which generated what attached the extension of ".log" to the test program name as this file name, and carried out [above-mentioned] capture processing is written in. An example of these contents was shown as a halt part (12) of a test program in drawing 10 .

[0059] It is the processing which obtains a symbol table from the load module of the test program (233) under activation, and acquires the data name inside a test program (233) at step 1722. With the gestalt of this operation, the information on the symbol used inside the test program of the format shown in drawing 7 is acquired to a file in the load module of the test program on the storage section (110) using the nm command. Next, in this file, the identifier of a symbol extracts the thing showing data and, finally acquires the variable name inside a test program.

[0060] At step 1723, the automation procedure and the screen capture of step 1200 publication on a virtual terminal (130) are performed, the value in the condition that the test program detected the error and has stopped about the variable inside a test program (233) is acquired, and it records as fault information (111). After transmitting a character string to a virtual terminal in the variable name inside the test program obtained by processing of step 1722 and displaying a data value on a user level debugger, the screen capture of the output of a user level debugger is carried out, and, specifically, it records on the file which describes the fault information created at step 1721.

[0061] For example, if screen rolling of what the user level debugger outputted with the display procedure (line feed) of a request variable, the value (line feed) of a request variable, and the prompt (line feed) is carried out on a 24 line x80 figure virtual terminal, a prompt will be displayed on the display procedure of a request variable by the 22nd line, and will be displayed on the 23rd line by the value of a request variable, and the 24th line. It enables them to obtain these as a character string and to carry out program manipulation by the screen capture function. The fault information acquired at a step here is the display procedure (113) of a request variable, and the value (114) of a request variable in drawing 10 .

[0062] The information display which is useful to analyzing the flow at the time of test program activation, such as back trace of the stack of the stopped test program (233), in the automation procedure of the step 1200 publication on a virtual terminal (130) is made to perform to a user level debugger at step 1724. In addition, this processing is functional dependence of the user level debugger to be used.

[0063] For example, in performing bag trace, the character string "bt (return)" is transmitted to a virtual terminal, and directions are given to a user level debugger. Thereby, a user level debugger analyzes the stack frame of a stopped test program. Consequently, a function name including the stopped point in which it is shown in what kind of career the current halt point has been called, and the source file name containing that function are acquired.

[0064] Furthermore, it acquires by the screen capture which performs a result on a virtual terminal as a result of a user level debugger, and leaves as fault information (111) like processing at step 1723. These showed as an analysis result (116) by the directions (115) to the user level debugger in drawing 10 , and the user level debugger.
[0065] What is necessary is that relate with the file which it is at the processing initiation time of step 1721, and turns on this logging function and stores the fault information (111) on secondary storage, and are at the processing termination time of step 1724, are turning off a logging function, and the automatic dialogue actuation to a user level debugger comes to carry out only in a key type when the log output function to a file is in a virtual terminal, although the variable value of the test program obtained from a user level debugger acquired by the screen capture function with the gestalt of this operation.
[0066] Thus, after this software verification equipment stops by the break point which the test program set up, it is characterized by having obtained from the symbol table in which the load module of a test program includes the variable name which the test program uses, and having collection processing of the fault information which acquires the variable value of the test program under halt through the user level debugger who is operating the test program.
[0067] With the gestalt 3 of gestalt 3. implementation of operation, after stopping by the break point to which the test program was set, the collection processor of the fault information which acquires the value of the variable which OS of a test objective uses is realized.
[0068] Drawing 11 expresses the structure-of-a-system Fig. which realizes the gestalt 3 of operation. With the gestalt of this operation, it has composition on a virtual terminal (150) with the communication facility which operates on OS verification equipment (100) which can debug OS on a target machine (200).
[0069] 150 is a virtual terminal which operates on OS verification equipment. This virtual terminal connects with the kernel level debugger on a target machine in order to carry out debugging actuation of the OS of the test objective on [from OS verification equipment (100)] a target machine using communication facility. In addition, as for the I/O to this virtual terminal (150), the fault information collection section (140) hits on OS verification equipment (100).
[0070] 221 is a kernel level debugger and is realized in OS of a test objective. The kernel level debugger is related with the specific communication link port on a target machine, and it is outputting and inputting using this communication link port, and it is possible to carry out debugging actuation of the OS of a test objective.
[0071] 7 is the load module of OS of a test objective. 8 is a file (it is also only called "the symbol information used inside OS of a test objective") showing the symbol information used inside OS of a test objective. The contents are shown in drawing 12 .
[0072] Drawing 12 shows the file (8) showing the symbol information used inside OS of a test objective, and its format. Symbol information is information which shows the address with which the data and the function name (symbol) which are used inside OS are assigned, when OS is developed on memory. Usually, after compiling the source file of a program, the address is decided, when linked. Symbol information is included in the load module. Moreover, symbol information can be outputted to a file etc. at the time of link processing.
[0073] For example, in UNIX, the symbol information on OS is in the condition that a

symbol table is contained in the load module of OS, and is acquired by setting and performing the pass of the load module of OS to the argument of the nm command. Moreover, this can require creation of OS generate time. With the gestalt of this operation, it treats about the approach of acquiring from the load module of OS.
[0074] 81 expresses the address of a symbol. 82 expresses the identifier of a symbol. The symbol which is data is distinguished for information here. 83 expresses a symbol name. When the class of symbol is data, it is the identifier of a variable data. The variable data obtained from this symbol information serves as an object which collects the values in that time as fault information, when a test program stops by the break point.
[0075] Drawing 13 expresses the flow of processing of the fault information collection section in the gestalt of this operation. Processing of the fault information collection section expressed to drawing 9 flowed, and drawing 3 was boiled, in addition extended step 1725 and step 1726. **** in the step extended to below -- it ****** just. It is the processing which obtains the address with which the data name within OS and it were assigned at step 1725 from the symbol information (8) used inside the load module of OS of a test objective. In the information shown in drawing 12 , the symbol name (83) in which the address (81) is included in data space is obtained by the following approaches. In the load module (7) of OS of the test objective on the storage section (110), the information on the symbol used inside OS of the test objective of the format shown in drawing 12 is acquired to a file using the nm command.
[0076] At step 1726, the value in the condition that the test program (233) detected the error and has stopped about the data which OS (220) of a test objective is using in the automation procedure of the step 1200 publication performed on a virtual terminal (150) is displayed, an output is acquired by the screen capture, and it records as fault information (111).
[0077] After adding a postscript to the file which describes the fault information (111) which specifically created the information on the symbol inside OS of the test objective obtained by processing of step 1725 at step 1721 (8 in drawing 14), A screen capture is carried out after performing continuously actuation which dumps the data space of a kernel on a kernel level debugger with the key type to a virtual terminal (150). A capture is carried out and data are added to the file which describes fault information (111) (output by the demand (17) to the kernel level debugger in drawing 14 , and the kernel level debugger (18)). It becomes possible to get to know which variable of OS the contents of the memory of the specific address are actually assigned using the information acquired here. If it forces, a test program can know the variable value inside OS at the time of failure detection.
[0078] Although the data value inside OS of the test objective obtained from a kernel level debugger was acquired by the screen capture function with the gestalt of this operation When the log output function to a file is in a virtual terminal In being at the processing initiation time of step 1725, relating with the file which turns on this logging function and stores the fault information (111) on secondary storage, being at the processing termination time of step 1726, and turning off a logging function Only a key type comes to carry out automatic dialogue actuation to a virtual terminal (150).
[0079] Thus, this software verification equipment is characterized by having obtained from the symbol table in which the load module of OS includes the variable name which OS of a test objective uses, and having collection processing of the fault information

which acquires the value which the variable inside OS under the situation of having been examined by the test program holds through a kernel level debugger, when it stops by the break point which the test program set up.
[0080] With the gestalt 4 of gestalt 4. implementation of operation, after stopping by the break point to which the test program was set, the collection processor of fault information which acquires the condition of the resource which OS which a test program uses manages is realized.
[0081] Drawing 15 expresses the structure-of-a-system Fig. which realizes the gestalt 4 of operation. 241 is a command interpreter for failures used in case the fault information which operates on a target machine (200) is collected. At the time of test initiation, the command interpreter for these failures is generated by the demand from OS verification equipment (100), and resides permanently on a target machine during test implementation. 160 is the virtual terminal which operates on OS verification equipment, and is connected with the group (240) of the program started from the command interpreter (241) for failures which operates on a target machine by communication facility. As for the I/O to this virtual terminal, the fault information collection section (140) hits on OS verification equipment (100). 240 is the group of the program which the command interpreter (241) for failures which operates on OS (220) of a test objective starts on a target machine (200). Various programs (242) are generated on a target machine (200) by the demand to the command interpreter (241) for failures which the fault information collection section (140) performs through a virtual terminal (160), and the standard output of these programs can check on a virtual terminal (160) by it. 140 is the fault information collection section. Processing of the fault information collection section (140) is extended in this example. For details, it is shown in drawing 18 .
[0082] Drawing 16 expresses the file (5) and an example of a format of the definition about activation of a test program. The element extended compared with the file (5) of the definition expressed to drawing 4 is as follows. 55 shows the automatic dialogue procedure (the contents of the processing of an operator of a key type) carried out on the virtual terminal connected with the command interpreter on a target machine. In this example, the character strings (ps -aluminum in drawing 16 etc.) and return code which were described after the identifier "CMD:" are sent to a virtual terminal.
[0083] Drawing 17 shows processing of the trial automatic activation-ized section (120) of the gestalt of this operation. Drawing 17 extended processing of step 900 to processing of the trial automatic activation-ized section (120) expressed to drawing 5 . The step extended to below is explained. At step 910, a virtual terminal (160) is prepared on OS verification equipment (100), and a command prompt is generated on a target machine (200) using the communication facility of a virtual terminal. With OS verification equipment, through the actuation on this virtual terminal (160), the various commands of various programs (242) are started on a target machine (200), and it is possible to obtain that result.
[0084] Drawing 18 expresses processing of the fault information collection section (140) of the gestalt of this operation. Drawing 18 extended step 1727 and step 1728 to processing of the fault information collection section (140) expressed to drawing 13 . Processing at the step extended to below is explained.
[0085] At step 1727, in order to acquire information about the environment which has influence on activation of a test program, such as information on the secondary storage

(210) on a target machine, the procedure performed on the command interpreter (241) for failures on a target machine is acquired. With the gestalt of this operation, the character string which starts in an identifier "CMD:" is describing the procedure performed on the command interpreter (241) for failures on a target machine in the definition (5) about activation of the test program shown in drawing 16 .

[0086] The character string which expresses with step 1728 actuation of acquiring the condition of the resource which OS which the test program obtained by processing of step 1727 uses in the automation procedure indicated to step 1200 performed on a virtual terminal (160) manages is sent to the command interpreter (241) for failures. A command interpreter interprets this character string and performs directed actuation.

[0087] As a result of being obtained by this, an output carries out the screen capture of the virtual terminal (160), changes it into the character-string data (24 lines x 80 figures) according to virtual terminal size, and is recorded on a file as fault information (111). The fault information acquired at a step here was shown in drawing 19 as the directions (117) to a command interpreter, and a condition (118) of the resource which OS manages.

[0088] Although acquired with the gestalt of this operation by the screen capture which performs information about the test atmosphere acquired by actuation performed on the command interpreter (241) for failures in a virtual terminal (160) When the log output function to a file is in a virtual terminal In being at the processing initiation time of step 1728, turning on this logging function, relating a log file with the file which stores the fault information (111) on secondary storage, being at the processing termination time of step 1728, and turning off a logging function Only a key type comes to carry out automatic dialogue actuation to a virtual terminal (160).

[0089] Thus, this software verification equipment is characterized by to have collection processing of fault information of being the resource which a test program uses, performing actuation for acquiring the information about the execution environment of a test program which influences the activation result of a test program because processing of OS changes according to that condition on the command interpreter which operates on OS of a test objective, and obtaining this result after stopping by the break point which . the test program set up.

[0090] It makes it possible for the gestalt 5 of gestalt 5. implementation of operation to define management performed after a setup of the time-out time amount which enables a hang-up judging at a test program, and a test program hang-up in relation to claim 5. When the test program is not completed after time-out time amount progress, it judges with the test program having caused the hang-up, and the device which automates a series of processings of the management at the time of a test program halt, fault information collection, test program termination, and a hang-up is realized.

[0091] Drawing 20 expresses the definition (5) about activation of the test program with which the member was extended, in order to realize the gestalt 5 of operation. The member extended to below is explained. 56 is time-out time amount. Even if the time amount shown here passes, when a test program is not completed, it is judged that the test program caused the hang-up. With the gestalt of this operation, the number (10 in drawing 20 and instantiation) numerically written after the identifier "TOUT:" is treated as the number of seconds.

[0092] 57 is the management after judging with the test program having hung-up. With the gestalt of this operation, the character string and return code which were written after

the identifier "HUP:" are transmitted on a virtual terminal (160) in the automation procedure indicated to step 1200. Here, the processing which releases the resource which OS which a test program uses manages is registered (in the example of drawing 20 , temporary file (*. tmp) is eliminated with the rm command). This solves the problem produced by releasing the resource which OS manages by a test program not being completed to the last.

[0093] Drawing 21 expresses the flow of processing of the trial automatic activation-ized section (120). Drawing 21 extended processing of step 1550, step 1730, and step 1740 to processing of the trial automatic activation-ized section (120) expressed to drawing 17 , and changed processing of step 1600 into step 1601.

[0094] At step 1550, while acquiring the present time of day as test program start time, the alarm is set up so that a time-out may occur after the time-out time amount of the test program gained by processing of step 1300.

[0095] At step 1601, even if the test program is not completed, when the alarm set up at step 1550 is generated (the time-out arose), it moves to processing of step 1710.

[0096] At step 1730, processing is changed by whether the time-out occurred. When the time-out has occurred and it has not occurred to processing of step 1740, it progresses to processing of step 1700.

[0097] At step 1740, although it is processing when the time-out has occurred, it is shown in drawing 22 for details. Drawing 22 expresses the processing after a test program time-out extended to the trial automatic activation-ized section with the gestalt 5 of operation. The contents of the processing extended to below are explained.

[0098] At step 1741, it is the processing which stops a test program. For example, in the case where OS of a test objective is UNIX, a SIGINT signal is sent to the test program under operation on a user level debugger, and a test program is stopped. For example, the code of an interruption code (ctrl-c) is typed on a virtual terminal (130).

[0099] The fault information collection section (1620) processes at step 1720. The contents of processing have been shown in drawing 18 .

[0100] A user level debugger (232) is made to end a test program (233) at step 1742 in the automation procedure indicated to step 1200 on a virtual terminal (130). For example, the character string "quit (return)" is transmitted to a virtual terminal (130).

[0101] At step 1743, the management (57) approach after a test program hang-up is read from the definition (5) about activation of a test program. As shown in drawing 20 , as for the definition (5) about activation of a test program, the management after a test program hang-up (57) is added with the gestalt of this operation. In the example of drawing 20 , the character string "rm *.tmp" following the character string "HUP:" is read.

[0102] At step 1744, while transmitting the character string acquired by processing of step 1743, and a return code on a virtual terminal (160) in the automation procedure which indicated to step 1200, a test program records on the purport and fault information (111) which caused the hang-up and interrupted activation.

[0103] Thus, this software verification equipment is having enabled the definition of the time-out time amount of a test program, and the management after hang-up detection. Even if a test program goes through the time-out time amount of a test program, when not ending or stopping After judging that the test program caused the hang-up, transmitting a signal to a test program and stopping it on a user level debugger, while acquiring a halt location Fault information is collected, a test program is further terminated on a user level

debugger, and it is characterized by having the experimental automation approach of performing management after said hang-up detection.
[0104] With the gestalt 6 of gestalt 6. implementation of operation, in order to analyze the factor which prevented experimental automatic activation stopping by the hang-up of a test program, and caused the hang-up, in the function which carries out automatic collection of the data, the device in which time-out judging time amount is set up efficiently is realized.
[0105] Drawing 23 expresses the flow of processing of the trial automatic activation-ized section (120). In order to realize the gestalt 6 of operation, processing of step 1350 and step 1750 was extended to drawing 21 . Processing at the step extended to below is explained.
[0106] At step 1350, reopon of the definition file (5) about activation of a test program is carried out in the mode which can be written in. With the gestalt 6 of operation, it rewrites to the value carried out based on the time amount which activation of a test program actually took the time-out time amount (55) in the definition file (5) about activation of the test program extended with the gestalt 5 of operation, and (drawing 20), and the hang-up judging time amount at the time of next test implementation is optimized.
[0107] At step 1750, while acquiring the present time of day as test program end time, elapsed time +alpha from the test program start time acquired by processing of step 1550 is written out as time-out time amount (55) in the definition file (5) about activation of a test program. + alpha may be the time amount which sees allowances to a hang-up judging, for example, the elapsed time from the test program start time computed at step 1750 is sufficient as it.
[0108] thus, the time-out time amount decision of the test program which carries out time-out time amount of a test program based on the track record value at the time of test program operation, and specifies it in the software verification equipment and the approach of automating the trial of software -- it carries out.
[0109] With the gestalt 7 of gestalt 7. implementation of operation, in order to analyze the factor which prevented experimental automatic activation stopping by the abnormal stop of a test program, and caused the abnormal stop, the device which makes it possible to carry out automatic collection of the data is realized.
[0110] Drawing 24 is the definition (5) about activation of the test program which extended drawing 20 , in order to realize the gestalt 7 of operation. The part extended to below is explained. An escape is for processing continuation actuation on a user level debugger, when it stops in response to the signal which a test program assumes. 58 is the information about the signal with which a test program assumes reception. The line number which receives the signal with which a signal class:test program does the file-name:assumption of this format (initiation): It consists of a line number (termination) which receives the signal to assume.
[0111] With the gestalt of this operation, in the situation that 32 kinds of signals are defined for OS of a test objective by UNIX, the notation of a signal class is made the same as the format which a user level debugger displays, in case the test program under activation stops in response to a signal.
[0112] Although it is the signal (58) which the test program in drawing 24 assumes and the signal class has specifically become SIGUSR1, this is the same as the notation

obtained on a user level debugger. In the definition of the information about the signal (58) with which the test program shown in drawing 24 assumes reception, SIGUSR1 signal supposes that it may receive in the 100th [less than] line from the 1st line of a test_src1.c file.

[0113] Drawing 25 expresses processing of the trial automatic activation-ized section (120). In order to realize the gestalt 7 of operation, processing of step 1760, step 1761, and step 1765 was extended to drawing 23 . Processing at the step extended to below is explained.

[0114] At step 1760, when a test program carries out an abnormal stop, it progresses to step 1761 and progresses to step 1765 except it. Although the abnormal stop of the test program on a user level debugger is because the test program received the signal, this signal is sent according to being sent from OS, since the description error of a test program performed unjust processing, and the failure of OS. On the other hand, a halt which can assume a test program on a user level debugger is because the signal which can be assumed by program manipulation was received [having performed the break point or].

[0115] From this, decision of that the test program carried out the abnormal stop After carrying out the screen capture of the virtual terminal (130) and acquiring the condition that the test program has stopped, to the string array of terminal size (for example, 24 lines x 80 figures), by comparison processing of the information acquired as a character string A halt factor (which signal did you receive and the processing [which file] of the how many lines suspend at the time of activation?) is grasped, and it judges that a halt factor is not the signal (58) which not a break point but a test program assumes, either.

[0116] However, since the location stopped in response to a signal may be in the library function which is not contained in the source file of a test program, back trace actuation which was described in step 1724 is performed, and it judges in the final call location in a test program.

[0117] The capture of the virtual terminal screen is specifically carried out for back trace information. Terminal size After obtaining to the string array of (24 line x80 figure [for example,]), it is the analysis result of a stack frame. The line number in the file name which called the location which the test program has stopped, and its file When the signal which it is contained in the range in the signal (58) which a test program assumes, and was received by said halt factor, and the signal which a test program assumes are in agreement, it judges with it not being an abnormal stop.

[0118] At step 1765, when the test program has stopped in response to a signal by judgment processing of step 1760 by normal processing, it progresses to step 1766. When it ends except [its], it progresses to step 1750.

[0119] At step 1766, continuation actuation of the test program is carried out on a user level debugger. Specifically, automatic dialogue actuation of carrying out continuation activation of the test program on a user level debugger is performed to a virtual terminal (130). Then, it returns to processing of step 1601.

[0120] Although step 1761 is processing when a test program carries out an abnormal stop, it is shown in drawing 26 for details.

[0121] Next, the processing after the test program which extended the trial automatic activation-ized section (120) with the gestalt 7 of operation shown in drawing 26 carries out an abnormal stop is explained. Processing at the step extended to below with the

gestalt of this operation is explained. The fault information collection section (140) processes at step 1720. The contents have been shown in drawing 18 .
[0122] At step 1742, automation procedure indicated to step 1200 by the virtual terminal (130) is performed, and a test program (233) is terminated on a user level debugger (232). For example, the character string "quit (return)" is transmitted to a virtual terminal (130).
[0123] At step 1743, since the test program is not carried out to the last with the gestalt of this operation as well as the gestalt 5 of operation, it is the purpose which releases the resource which OS which the test program used manages, and the management (57) after a test program hangs-up is read from the definition (5) about activation of a test program.
[0124] At step 1762, in the automation procedure indicated to step 1200 to the virtual terminal (160), the character string acquired by processing of step 1743 is transmitted, and processing after a test program abnormal stop is carried out on the command interpreter (241) for failures. In addition, it records on the purport and fault information (111) in which the test program carried out the abnormal stop.
[0125] Thus, after a test program carries out the abnormal stop of the management after a test program abnormal stop in the experimental verification approach by having made the definition possible, while acquiring a halt location, it is characterized by automating the trial which collects fault information, is made to end a test program on a user level debugger, and performs management after said test program abnormal stop by the fault information collection section.
[0126] With the gestalt 8 of gestalt 8. implementation of operation, the device of the trial automatic activation which makes possible recovery after it produces a system down according to the serious failure of OS of a test objective and a trial goes wrong is realized.
[0127] Drawing 27 expresses OS verification equipment (100) and the target machine (200) which extended the function which resets the target system to drawing 15 , in order to realize the gestalt 8 of operation. Below, the extension is explained.
[0128] 250 is the reset circuit of hardware and has the function which resets a target machine by the signal input from the outside (contact-input port). 170 is the output signal control software, uses a contact output port and outputs the signal for resetting a target machine. In addition, in the low, connection of 170 and 250 is carried out electrically.
[0129] Drawing 28 is the definition (5) about activation of the test program which extended drawing 24 , in order to realize the gestalt 8 of operation. The part extended to below is explained.
[0130] 59 is reset postponement time amount. Even if time amount here passes, when a hang-up condition continues by the time amount for which it waits in case a test program is led to a halt after a judgment as the test program hung-up, hardware reset processing of a target machine is performed. With the gestalt of this operation, the number written in the figure after an identifier "RST:" is treated as the number of seconds.
[0131] Drawing 29 is processing of the fault information collection section (140), and in order to realize the gestalt 8 of operation, it is extending the processing of the fault information collection section (140) shown in drawing 22 . The processing in the step extended to below is explained.
[0132] Step 1770 changes processing by whether the test program stopped with the signal sent by processing of step 1741. When it stops, it moves to processing of step 1720, and when not stopping, it moves to processing of extended step 1771.

[0133] At step 1771, the processing here in the test program judge that caused the hang-up is changing processing by the case of the 1st time, and the 2nd case. In the case of the 1st time, it moves to processing of step 1772 to the step 1775. It moves to processing of step 1780 to the step 900 including the processing which resets a target machine on the way the 2nd case. Below, processing when step 1771 is the 1st time is explained.
[0134] A kernel level debugger (221) is called at step 1772. Data transmitting processing which produces effectiveness equivalent to the specific key type specifically performed to a virtual terminal (150) in case an operator calls a kernel level debugger is performed.
[0135] At step 1773, processing is changed by the ability of the kernel level debugger (221) to have been called by processing of step 1772. This judgment is made by [as being the following]. The screen capture of the virtual terminal (150) is carried out, and alphabetic data is obtained to the string array according to the size of a virtual terminal. For example, it becomes the processing which incorporates the alphabetic character currently outputted to the array of 80 figure x24 line on the virtual terminal (130). It can judge whether the kernel level debugger was able to be called by whether the prompt of the kernel level debugger which the output of a kernel level debugger is displayed on a virtual terminal (130), and is displayed on the lowest line on a screen (the 24th line) in the situation that the contents of a display are scrolled upward was obtained.
[0136] When a kernel level debugger (221) is able to be called, it moves to processing of step 1774, and when a kernel level debugger is not able to be called, it judges that OS of a test objective has caused the hang-up, and moves to processing of step 1780.
[0137] A kernel level debugger (221) is ended and actuation of OS is made to continue at step 1774. Data transmitting processing which produces effectiveness equivalent to the specific key type specifically performed to a virtual terminal (150) in case an operator withdraws from a kernel level debugger is performed.
[0138] At step 1775, a part for the reset postponement time amount in the definition (5) about activation of a test program is waited. Then, it returns to processing of step 1670. Step 1671 is judged as the 2nd times next time at the time of passage.
[0139] Below, processing, i.e., processing of step 1680 to the step 1682 after the condition that a test program does not stop becomes certain, when step 1771 is the 2nd times is explained.
[0140] The output signal control software (170) is operated and the hardware reset circuit (250) of a target machine is made to reset delivery and a target machine for a reset signal at step 1780.
[0141] At step 1781, in order that a test program may not terminate normally, it leaves the purport which made the target machine reset to failure record (111).
[0142] At step 1782, it waits for a target machine to start.
[0143] In the situation made to call from the processing after the test program time-out after a test program produces a time-out with the gestalt 7 of operation (step 1740), even if processing of the fault information collection section explained above reboots a target machine because the target machine itself caused the hang-up, it can continue and perform the next trial shown in the pass (4) of a test program.
[0144] Thus, when it sets to the automation approach of a trial of software and OS of a test objective hung-up and crashes, after it sends out a hardware reset signal to the target machine with which OS of a test objective operates and a target machine reboots, it is characterized by being the automation approach of the trial which carries out continuation

activation of the trial from the experimental degree which caused the failure.
[0145] The gestalten 1-8 of the gestalt 9. above-mentioned implementation of operation explained OS verification equipment as an example as software verification equipment. However, to say nothing of being contained also when verifying an application program as software other than OS, the above-mentioned software verification equipment is a software program which operates on a computer also except the above, and contains ** which can be started with a test program.
[0146] Although the gestalten 1-8 of the gestalt 10. above-mentioned implementation of operation explained the case where software verification equipment and a target machine (200) were realized on another computer, as shown in drawing 30 as an example, the function with which software verification equipment was equipped, for example, the pass information on a software program group (1) and a test program, (4) may be installed on a target machine (200).
[0147]
[Effect of the Invention] As mentioned above, since according to the software verification equipment or the approach concerning this invention it means that the test program stopped with as when the break point set up on the test program is passed, it can carry out to having made freely the condition that the failure had arisen hold by setting a break point as the location which detected the failure.
[0148] Moreover, according to this invention, the internal state of a test program can be acquired in the condition [that the test program has stopped after the test program detected the failure]. Thereby, the information for specifying the cause of a failure at the time of failure detection can be acquired about a failure with difficult reappearance, and it can be said that a test objective program makes avoidable the problem judged to be a failure by the grasp of procedure from which a test objective program starts a failure, and the error of a test program.
[0149] Moreover, according to this invention, the internal state of OS's, such as a variable value about the resource inside OS which the test program used, can be acquired in the condition [that the test program has stopped after the test program detected the error]. For this reason, the information for analyzing the cause of a failure at the time of failure detection can be acquired about a failure with difficult reappearance.
[0150] Moreover, according to this invention, after a test program detects an error, in a condition [that the test program has stopped], it is effective in the ability to acquire the condition of the resource which OS which a test program uses manages.
[0151] Moreover, according to this invention, when a test program hangs-up, a test program is stopped, automatic collection of the data for analyzing the factor of a hang-up is carried out, a test program is terminated, and the resource which the test program used is released. Thereby, even when a test program hangs-up, data required for factor analysis can be acquired and there is effectiveness of the ability to make experimental automatic activation continue.
[0152] Moreover, according to this invention, in order to analyze the factor which prevented experimental automatic activation stopping by the hang-up of a test program, and caused the hang-up, in the function which carries out automatic collection of the data, it is effective in becoming possible to set up time-out judging time amount efficiently.
[0153] Moreover, according to this invention, it is effective in making it possible to carry out automatic collection of the data in order to analyze the factor which prevented

experimental automatic activation stopping by the abnormal stop of a test program, and caused the abnormal stop.

[0154] Moreover, according to this invention, after OS of a test objective resets in hardware the target machine with which OS of a test objective operates in the situation that a trial fails in a serious failure by the lifting and having carried out the system down and a target machine reboots by activation of a test program, it is effective in enabling trial automatic activation which carries out continuation activation from the trial next to the trial which caused the failure.

[Translation done.]

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]

[Drawing 1] Drawing showing an example of a system which realizes the gestalt 1 of operation.

[Drawing 2] Drawing showing an example of the folder according to trial item which manages a test program.

[Drawing 3] Drawing showing an example of the pass of a test program.

[Drawing 4] Drawing showing ****** of the definition about activation of a test program.

[Drawing 5] Drawing showing an example of the flow of processing of the trial automatic activation-ized section which realizes the gestalt 1 of operation.

[Drawing 6] Drawing showing an example of a system which realizes the gestalt 2 of operation.

[Drawing 7] Drawing showing an example of the information on the symbol used inside the test program.

[Drawing 8] Drawing showing an example of the flow of processing of the trial automatic activation-ized section which realizes the gestalt 2 of operation.

[Drawing 9] Drawing showing an example of processing of the fault information collection section in the gestalt 2 of operation.

[Drawing 10] Drawing showing an example of the fault information in the gestalt 2 of operation.

[Drawing 11] Drawing showing an example of a system which realizes the gestalt 3 of operation.

[Drawing 12] Drawing showing an example of the symbol used inside OS of a test objective.

[Drawing 13] Drawing showing an example of processing of the fault information collection section in the gestalt 3 of operation.

[Drawing 14] Drawing showing an example of the fault information in the gestalt 3 of operation.

[Drawing 15] Drawing showing an example of a system which realizes the gestalt 4 of operation.

[Drawing 16] Drawing showing an example of the definition about activation of the test

program extended with the gestalt 4 of operation.
[Drawing 17] Drawing showing an example of processing of the trial automatic activation-ized section extended with the gestalt 4 of operation.
[Drawing 18] Drawing showing an example of processing of the fault information collection section which realizes the gestalt 4 of operation.
[Drawing 19] Drawing showing an example of the fault information in the gestalt 4 of operation.
[Drawing 20] Drawing showing an example of the definition about activation of the test program which realizes the gestalt 5 of operation.
[Drawing 21] Drawing showing an example of processing of the trial automatic activation-ized section which realizes the gestalt 5 of operation.
[Drawing 22] Drawing showing an example of processing after the test program time-out in the trial automatic activation-ized section.
[Drawing 23] Drawing showing an example of processing of the trial automatic activation-ized section which realizes the gestalt 6 of operation.
[Drawing 24] Drawing showing an example of the definition about activation of the test program which realizes the gestalt 7 of operation.
[Drawing 25] Drawing showing an example of processing of the trial automatic activation-ized section which realizes the gestalt 7 of operation.
[Drawing 26] Drawing showing an example of processing after the test program abnormal stop which realizes the gestalt 7 of operation.
[Drawing 27] Drawing showing an example of OS verification equipment which realizes the gestalt 8 of operation.
[Drawing 28] Drawing showing an example of the definition about activation of the test program which realizes the gestalt 8 of operation.
[Drawing 29] Drawing showing an example of processing of the fault information collection section which realizes the gestalt 8 of operation.
[Drawing 30] Drawing showing an example of a system which realizes the gestalt 10 of operation.
[Drawing 31] Drawing showing an example of the structure of a system conventionally.
[Description of Notations]
1 Test Program Group, 1a Folder for Every Trial Item, 2 Src Folder, 3 bin, 4 The pass information on a test program, 5 The definition about activation of a test program, 6 The file, 7 showing symbol information The load module of OS of a test objective, 8 The symbol information, 10 which are used inside OS of a test objective Program test equipment, 10a A test script management tool, 10b Test automation control means, 10c A test script activation means, 10d Test yes-no decision means, 10e program failure specification means, 10f Test script input means, 10g A test-result output means, 10h Mode change / activation means, 10i A debugging command input means, 20 Target system, 21 The source file of a test program, 22 The file which described the procedure which generates a test program, 30 A tester, 31 The load module of a test program, 41 Pass of a test program, 51 argument information, 52 The information about a break point, 53 Function name, 54 A line number, 56 Time-out time amount, 57 At the time of a hang-up, management, 59 Reset postponement time amount, 61 The address of a symbol, 62 Symbol identifier, 63 A symbol name, 100 OS verification equipment, 101 Interface, 110 The storage section (test program storage section), 111 Fault information, 120 The

trial automatic activation-ized section, 130 A virtual terminal and 140 Fault information collection section, 150 A virtual terminal, 160 A virtual terminal, 200 Target machine, 201 An interface, 210 Secondary storage, 220 OS of a test objective, 221 The group of a kernel level debugger and 230 test programs, 231 A command interpreter, 232 A user level debugger, 233 A test program, 240 The group of a program, 241 A command interpreter, 242 Various programs, 300 Channel.

[Translation done.]

CLAIMS

[Claim(s)]

[Claim 1] It is software verification equipment which can start DEBAKKA which examines software. The test program storage section which memorizes the test program which examines the above-mentioned software using the application programming interface (henceforth "API") of the software made into a test objective, Specify the test program memorized by the above-mentioned test program storage section as a program operated using DEBAKKA, and above-mentioned DEBAKKA is started. A break point is set to the part of a test program where the specified test program grasps an error by API of the above-mentioned software using started DEBAKKA. Software verification equipment characterized by having the trial automatic activation-ized section performed using DEBAKKA which had the set-up test program started.

[Claim 2] The above-mentioned test program includes the error detection point which grasps that actuation of software is unusual by the activation result defined by API of the above-mentioned software. The above-mentioned test program storage section memorizes the definition about activation of a test program including the error detection point of a test program further. The above-mentioned trial automatic activation-ized section Software verification equipment according to claim 1 characterized by setting a break point as a test program based on the error detection point of the test program memorized by the above-mentioned test program storage section.

[Claim 3] The above-mentioned software verification equipment is software verification equipment according to claim 1 characterized by having the fault information collection section which collects the fault information at the time of a test program stopping when a test program stops further by the break point by which a setup was carried out [above-mentioned].

[Claim 4] The above-mentioned fault information collection section is software verification equipment according to claim 3 characterized by collecting the values of the variable of a test program based on the variable information which extracted and extracted the above-mentioned variable information using DEBAKKA from the above-mentioned test program including the variable information as information that the variable which uses the above-mentioned test program while a test program performs is specified.

[Claim 5] The above-mentioned fault information collection section is software verification equipment according to claim 3 characterized by collecting the values of the

variable of a test program based on the test objective variable information that the above-mentioned test objective variable information was extracted and extracted, using DEBAKKA from the above-mentioned software including the test objective variable information as information that the variable which uses the above-mentioned software while software performs is specified.

[Claim 6] The above-mentioned fault information collection section is software verification equipment according to claim 3 which changes the execution environment of the above-mentioned software and is characterized by acquiring the activation result of the above-mentioned software performed by performing the above-mentioned software on the changed execution environment.

[Claim 7] claims 3-6 characterized by collecting fault information while the above-mentioned trial automatic activation-ized section stops a test program using above-mentioned DEBAKKA when a test program is not completed after predetermined time amount progress, and the above-mentioned fault information collection section acquires the halt location of the stopped test program -- software verification equipment given in either.

[Claim 8] The above-mentioned trial automatic operation-ized section is software verification equipment according to claim 7 characterized by changing the above-mentioned predetermined time amount based on the activation result of a test program.

[Claim 9] claims 3-6 characterized by collecting fault information while the test program which detected that the test program carried out the abnormal stop of the above-mentioned trial automatic activation-ized section, and was detected using DEBAKKA is forced to terminate and the above-mentioned fault information collection section acquires the forced-termination location of the test program forced to terminate -- software verification equipment given in either.

[Claim 10] The above-mentioned trial automatic activation-ized section is software verification equipment according to claim 1 which sends out the signal which directs reset of the computer by which software operates after a test program carries out an abnormal stop, and is characterized by performing a test program after the above-mentioned computer reboots with the sent-out signal.

[Claim 11] The above-mentioned software is software verification equipment according to claim 1 characterized by including either of an operating system or an application program.

[Claim 12] It is the software verification approach which can start DEBAKKA which examines software. The test program storage process of memorizing the test program which examines the above-mentioned software using the application programming interface (henceforth "API") of the software made into a test objective, Specify the test program memorized at the above-mentioned test program storage process as a program operated using DEBAKKA, and above-mentioned DEBAKKA is started. A break point is set to the part of a test program where the specified test program grasps an error by API of the above-mentioned software using started DEBAKKA. The software verification approach characterized by having a trial automatic activation chemically-modified [which is performed using DEBAKKA which had the set-up test program started] degree.

[Translation done.]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号
特開2001－243089
（P2001－243089A）
(43)公開日　平成13年９月７日(2001.9.7)

(51)Int.Cl.'	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　11/28	３４０	Ｇ０６Ｆ　11/28	３４０Ａ　５Ｂ０４２

審査請求　未請求　請求項の数12　ＯＬ　（全 22 頁）

(21)出願番号　特願2000－49146(P2000－49146)

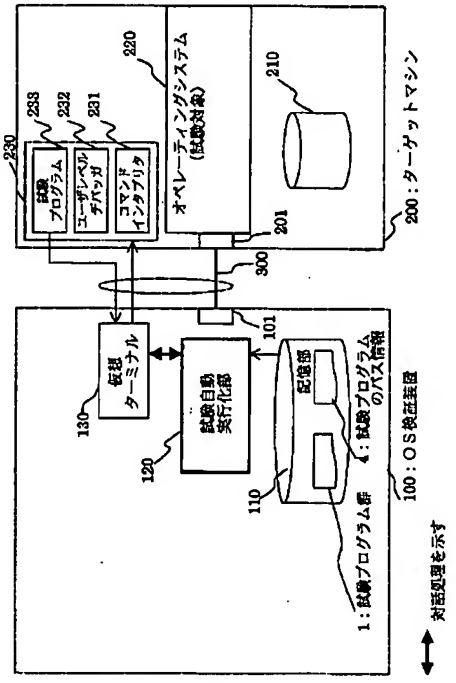
(22)出願日　平成12年２月25日(2000.2.25)

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号
(72)発明者　馬場　儀之
東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内
(74)代理人　100099461
弁理士　溝井　章司　（外２名）

Ｆターム(参考)　5B042 GA21 HH01 HH11 HH25 HH49 JJ23 KK06 KK14

(54)【発明の名称】　ソフトウェア検証装置及びソフトウェア検証方法

(57)【要約】
【課題】　試験対象ソフトウェアの異常の原因追求を容易にする検査装置を提供する。
【解決手段】　ソフトウェア検証装置は、試験対象とするソフトウェアのＡＰＩを用いて、上記ソフトウェアの試験を行う試験プログラムを記憶する試験プログラム記憶部１１０と、上記試験プログラム記憶部１１０に記憶された試験プログラムを、デバッカを用いて動作させるプログラムとして指定して上記デバッカを起動し、起動されたデバッカを用いて、指定された試験プログラムが上記ソフトウェアのＡＰＩによってエラーを把握する試験プログラムの箇所へブレイクポイントを設定し、設定した試験プログラムを起動されたデバッカを用いて実行する試験自動実行化部１２０とを備える。



1

【特許請求の範囲】

【請求項1】　ソフトウェアを試験するデバッカを起動可能なソフトウェア検証装置であって、

試験対象とするソフトウェアのアプリケーション・プログラミング・インタフェース（以下、「ＡＰＩ」という）を用いて、上記ソフトウェアの試験を行う試験プログラムを記憶する試験プログラム記憶部と、

上記試験プログラム記憶部に記憶された試験プログラムを、デバッカを用いて動作させるプログラムとして指定して上記デバッカを起動し、起動されたデバッカを用いて、指定された試験プログラムが上記ソフトウェアのＡＰＩによってエラーを把握する試験プログラムの箇所へブレイクポイントを設定し、設定した試験プログラムを起動されたデバッカを用いて実行する試験自動実行化部とを備えたことを特徴とするソフトウェア検証装置。

【請求項2】　上記試験プログラムは、上記ソフトウェアのＡＰＩで定義された実行結果によってソフトウェアの動作が異常であることを把握するエラー検出ポイントを含み、

上記試験プログラム記憶部は、さらに、試験プログラムのエラー検出ポイントを含む試験プログラムの実行に関する定義を記憶し、

上記試験自動実行化部は、上記試験プログラム記憶部に記憶された試験プログラムのエラー検出ポイントに基づいて、試験プログラムにブレイクポイントを設定することを特徴とする請求項1記載のソフトウェア検証装置。

【請求項3】　上記ソフトウェア検証装置は、さらに、上記設定されたブレイクポイントで試験プログラムが停止した場合、試験プログラムが停止した時点の障害情報を収集する障害情報収集部を備えたことを特徴とする請求項1記載のソフトウェア検証装置。

【請求項4】　上記試験プログラムは、試験プログラムが実行中に使用する変数を特定する情報としての変数情報を含み、

上記障害情報収集部は、デバッカを用いて、上記試験プログラムから上記変数情報を抽出し、抽出した変数情報に基づいて、試験プログラムの変数の値を収集することを特徴とする請求項3記載のソフトウェア検証装置。

【請求項5】　上記ソフトウェアは、ソフトウェアが実行中に使用する変数を特定する情報としての試験対象変数情報を含み、

上記障害情報収集部は、デバッカを用いて、上記ソフトウェアから上記試験対象変数情報を抽出し、抽出した試験対象変数情報に基づいて、試験プログラムの変数の値を収集することを特徴とする請求項3記載のソフトウェア検証装置。

【請求項6】　上記障害情報収集部は、上記ソフトウェアの実行環境を変更し、変更した実行環境上で上記ソフトウェアを実行させ、実行させた上記ソフトウェアの実行結果を取得することを特徴とする請求項3記載のソフトウェア検証装置。

2

【請求項7】　上記試験自動実行化部は、所定の時間経過後に試験プログラムが終了しない場合、上記デバッカを用いて試験プログラムを停止させ、

上記障害情報収集部は、停止させた試験プログラムの停止位置を取得するとともに、障害情報を収集することを特徴とする請求項3から6いずれかに記載のソフトウェア検証装置。

【請求項8】　上記試験自動実施化部は、試験プログラムの実行結果に基づいて、上記所定の時間を変更することを特徴とする請求項7記載のソフトウェア検証装置。

【請求項9】　上記試験自動実行化部は、試験プログラムが異常停止したことを検知し、デバッカを用いて検知した試験プログラムを強制終了させ、

上記障害情報収集部は、強制終了させた試験プログラムの強制終了位置を取得するとともに、障害情報を収集することを特徴とする請求項3から6いずれかに記載のソフトウェア検証装置。

【請求項10】　上記試験自動実行化部は、試験プログラムが異常停止した後に、ソフトウェアが動作する計算機のリセットを指示する信号を送出し、送出した信号によって上記計算機が再起動した後に、試験プログラムを実行させることを特徴とする請求項1記載のソフトウェア検証装置。

【請求項11】　上記ソフトウェアは、オペレーティングシステムまたはアプリケーションプログラムとのいずれかを含むことを特徴とする請求項1記載のソフトウェア検証装置。

【請求項12】　ソフトウェアを試験するデバッカを起動可能なソフトウェア検証方法であって、

試験対象とするソフトウェアのアプリケーション・プログラミング・インタフェース（以下、「ＡＰＩ」という）を用いて、上記ソフトウェアの試験を行う試験プログラムを記憶する試験プログラム記憶工程と、

上記試験プログラム記憶工程で記憶された試験プログラムを、デバッカを用いて動作させるプログラムとして指定して上記デバッカを起動し、起動されたデバッカを用いて、指定された試験プログラムが上記ソフトウェアのＡＰＩによってエラーを把握する試験プログラムの箇所へブレイクポイントを設定し、設定した試験プログラムを起動されたデバッカを用いて実行する試験自動実行化工程とを備えたことを特徴とするソフトウェア検証方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、計算機のＯＳ（Ｏｐｅｒａｔｉｏｎ　Ｓｙｓｔｅｍ）を試験する装置及び方法に関する技術である。特に、ＯＳが提供するアプリケーション・プログラミング・インタフェースを用いてＯＳのサービス内容を確認する検証装置において、試験

失敗時の原因追求を容易とさせる、試験自動実行方法を提供する。

【0002】

【従来の技術】プログラムのテストを自動化するための従来方法には、例えば、特開平8-305609号公報に開示されているテスト方法及びプログラムテスト装置がある。図31に、従来システムの構成の一例を示す。図31は、プログラムテスト装置10と、テスト対象のプログラムを実行する手段としてのターゲットシステム20と、検査者30、例えば、オペレータとの関係を示す。プログラムテスト装置は、テストスクリプト管理手段10aと、テスト自動化制御手段10bと、テストスクリプト実行手段10cと、テスト合否判定手段10dと、プログラム障害特定手段10eとを具える。さらにこのプログラムテスト装置10は、検査者30との間のインタフェース手段として、テストスクリプト入力手段10fと、テスト結果出力手段10gと、モード切変え/実行手段10hと、デバッグコマンド入力手段10iとを具える。これらインタフェース手段10f～10iは、それぞれ、テスト自動化制御手段10bと接続してある。

【0003】従来技術において、プログラムテスト動作は以下の通りとなっている。テスト対象プログラムに障害があるか否かの判定に好適なテスト対象プログラムにおける所定ステップでの実行結果を出力させる旨及び該実行結果に対するテスト項目を、少なくとも含むテストスクリプトを用意しておき、テスト対象のプログラムを実行すると共に、所定ステップでの前記テストスクリプトの処理により、テスト対象プログラムに障害があるかを判定する。

【0004】更に、従来技術において、障害特定動作がある。これは、以下のように行われる。検査者は、障害を含んでいたためプログラムテスト動作において不合格とされたプログラムに対する障害有無判定用テストスクリプトを、テストスクリプト入力手段10fを利用して、プログラムテスト装置10の表示装置に表示させる。次に、この障害を含むプログラムの障害箇所特定に好適な障害特定用テストスクリプトを、テストスクリプト入力手段10fを利用して、テストスクリプト管理手段10aの記憶部のファイルに登録する。

【0005】この登録する障害特定用テストスクリプトは、テストモード（上記公報からの引用省略）で障害があると判定された際に、得られたテスト結果から見て当該障害の特定に好適なように、障害有無判定用テストスクリプトを変更して作成する。例えば、ブレイクポイントやダンプすべきデータ名を変更している。この後、テストプログラムの一部或は全部を再実行する。結果として、障害特定用テストスクリプトの処理によって、障害箇所を特定できるとしたものであった。

【0006】

【発明が解決しようとする課題】しかしながら、従来例のように、一旦障害を検出した後に改めて障害箇所を特定するスクリプトを登録して再実行する試験方法では、再現性が低い障害に対して、原因追求が困難であるという課題があった。この発明は、上述のような課題を解決するためになされたもので、第1の目的は、試験対象プログラムを試験プログラムを用いて自動的に検証する装置において、試験プログラム上にブレイクポイントを設定することを可能とし、試験動作を停止させるものである。

【0007】第2の目的は、第1の目的で設定するブレイクポイントを試験プログラムが障害を検出する場所を対象にする状況で、試験プログラム停止後に試験プログラムの内部状態を得ることによって、再現が困難な障害について、障害検出時に障害原因を解析するための情報を得るものである。

【0008】第3の目的は、第1の目的で設定するブレイクポイントを試験プログラムが障害を検出した場所を対象にする状況で、試験プログラム停止後に試験対象プログラムの内部状態を得ることによって、再現が困難な障害について、障害検出時に障害原因を解析するための情報を得るものである。

【0009】第4の目的は、第1の目的で設定するブレイクポイントを試験プログラムが障害を検出した場所を対象にする状況で、試験プログラムが使用する資源で、その状態によりオペレーティングシステム（以下、OSと記す）の処理が変わることで試験プログラムの実行結果に影響するような試験プログラムの実行環境に関する情報を得るものである。

【0010】第5の目的は、タイムアウト判定時間を定義し、この時間内に試験プログラムが終了しない場合に試験プログラムがハングアップを起こしたと判断し、ハングアップを起こした要因を解析するためのデータを自動収集した後、試験プログラムを終了させ、試験動作の連続処理を可能にするものである。

【0011】第6の目的は、第5の目的において、効率良くタイムアウト判定時間を設定することを可能にするものである。

【0012】第7の目的は、試験プログラムの異常停止によって試験の自動実行が停止することを防ぎ、且つ異常停止を起こした要因を解析するためのデータ収集を行うものである。

【0013】第8の目的は、試験プログラムの実行によって試験対象のOSが重大障害を起こし、システムダウンしたことで試験が失敗する状況において、試験対象のOSが動作するターゲットマシンをハードウェア的にリセットし、ターゲットマシンが再起動した後に、試験動作の連続処理を可能にするものである。

【0014】

【課題を解決するための手段】この発明に係るソフトウ

ェア検証装置は、ソフトウェアを試験するデバッカを起動可能なソフトウェア検証装置であって、試験対象とするソフトウェアのアプリケーション・プログラミング・インタフェース（以下、「ＡＰＩ」という）を用いて、上記ソフトウェアの試験を行う試験プログラムを記憶する試験プログラム記憶部と、上記試験プログラム記憶部に記憶された試験プログラムを、デバッカを用いて動作させるプログラムとして指定して上記デバッカを起動し、起動されたデバッカを用いて、指定された試験プログラムが上記ソフトウェアのＡＰＩによってエラーを把握する試験プログラムの箇所へブレイクポイントを設定し、設定した試験プログラムを起動されたデバッカを用いて実行する試験自動実行化部とを備えたことを特徴とする。

【００１５】上記試験プログラムは、上記ソフトウェアのＡＰＩで定義された実行結果によってソフトウェアの動作が異常であることを把握するエラー検出ポイントを含み、上記試験プログラム記憶部は、さらに、試験プログラムのエラー検出ポイントを含む試験プログラムの実行に関する定義を記憶し、上記試験自動実行化部は、上記試験プログラム記憶部に記憶された試験プログラムのエラー検出ポイントに基づいて、試験プログラムにブレイクポイントを設定することを特徴とする。

【００１６】上記ソフトウェア検証装置は、さらに、上記設定されたブレイクポイントで試験プログラムが停止した場合、試験プログラムが停止した時点の障害情報を収集する障害情報収集部を備えたことを特徴とする。

【００１７】上記試験プログラムは、試験プログラムが実行中に使用する変数を特定する情報としての変数情報を含み、上記障害情報収集部は、デバッカを用いて、上記試験プログラムから上記変数情報を抽出し、抽出した変数情報に基づいて、試験プログラムの変数の値を収集することを特徴とする。

【００１８】上記ソフトウェアは、ソフトウェアが実行中に使用する変数を特定する情報としての試験対象変数情報を含み、上記障害情報収集部は、デバッカを用いて、上記ソフトウェアから上記試験対象変数情報を抽出し、抽出した試験対象変数情報に基づいて、試験プログラムの変数の値を収集することを特徴とする。

【００１９】上記障害情報収集部は、上記ソフトウェアの実行環境を変更し、変更した実行環境上で上記ソフトウェアを実行させ、実行させた上記ソフトウェアの実行結果を取得することを特徴とする。

【００２０】上記試験自動実行化部は、所定の時間経過後に試験プログラムが終了しない場合、上記デバッカを用いて試験プログラムを停止させ、上記障害情報収集部は、停止させた試験プログラムの停止位置を取得するとともに、障害情報を収集することを特徴とする。

【００２１】上記試験自動実施化部は、試験プログラムの実行結果に基づいて、上記所定の時間を変更することを特徴とする。

【００２２】上記試験自動実行化部は、試験プログラムが異常停止したことを検知し、デバッカを用いて検知した試験プログラムを強制終了させ、上記障害情報収集部は、強制終了させた試験プログラムの強制終了位置を取得するとともに、障害情報を収集することを特徴とする。

【００２３】上記試験自動実行化部は、試験プログラムが異常停止した後に、ソフトウェアが動作する計算機のリセットを指示する信号を送出し、送出した信号によって上記計算機が再起動した後に、試験プログラムを実行させることを特徴とする。

【００２４】上記ソフトウェアは、オペレーティングシステムまたはアプリケーションプログラムとのいずれかを含むことを特徴とする。

【００２５】この発明に係るソフトウェア検証方法は、ソフトウェアを試験するデバッカを起動可能なソフトウェア検証方法であって、試験対象とするソフトウェアのアプリケーション・プログラミング・インタフェース（以下、「ＡＰＩ」という）を用いて、上記ソフトウェアの試験を行う試験プログラムを記憶する試験プログラム記憶工程と、上記試験プログラム記憶工程で記憶された試験プログラムを、デバッカを用いて動作させるプログラムとして指定して上記デバッカを起動し、起動されたデバッカを用いて、指定された試験プログラムが上記ソフトウェアのＡＰＩによってエラーを把握する試験プログラムの箇所へブレイクポイントを設定し、設定した試験プログラムを起動されたデバッカを用いて実行する試験自動実行化工程とを備えたことを特徴とする。

【００２６】

【発明の実施の形態】実施の形態１．実施の形態１では、ＯＳを試験プログラムの実行によって検証する状況において、試験プログラムが障害検出直後に試験プログラムを停止させ、障害検出時の状態を保持させたままとする機構を提供する。そこで、この実施の形態は、ソフトウェア検証装置としてＯＳを検証するＯＳ検証装置を一例として説明する。また、実施の形態２から実施の形態８においても同様に、ＯＳ検証装置を一例として説明する。従って、試験対象となる試験対象プログラムは、ＯＳとなる。

【００２７】図１は、実施の形態１を実現するシステムの一例の構成図を表わしている。１００は、ソフトウェア検証装置の一例として、ＯＳ検証装置を表わしている。ＯＳ検証装置（１００）は、本発明において、ターゲットマシン上で行うＯＳの試験を支援する。本実施の形態では、ＯＳ検証装置（１００）は、記憶部（試験プログラム記憶部）（１１０）、試験自動実行化部（１２０）、仮想ターミナル（１３０）、ターゲットマシンと接続する通信路とのインタフェース（１０１）で構成される。

【００２８】１１０は、試験プログラム記憶部としての記憶部（２次記憶装置）である。ここには、ターゲットマシン上で実施する試験プログラム群（１）と、試験プログラムの２次記憶装置上の位置情報（パス）を表わした試験プログラムのパス情報を表わすファイル（単に、「試験プログラムのパス情報」ともいう）（４）がある。詳細は、図２と図３に示す。１２０は、ＯＳ検証装置における試験自動実行化部で、図５に示す動作を行う。１３０は、仮想ターミナルで、通信機能によって、試験対象のＯＳ上で動作する試験プログラムを動作させるセッションとしての試験プログラムのグループ（２３０）と接続する。仮想ターミナル（１３０）は、試験自動実行化部（１２０）からの指示を試験プログラムのグループ（２３０）へ通知する。

【００２９】２００は、ターゲットマシンを表わしている。この上で、試験対象のＯＳ（２２０）が動作する。２０１は、ＯＳ検証装置と接続する通信路とのインタフェースである。２１０は、ターゲットマシン上の２次記憶装置である。２２０は、試験対象のＯＳ（ＯＳプログラム）である。２３０は、試験対象のＯＳ（２２０）上で動作する試験プログラムのグループである。２３１は、試験対象のＯＳ上で動作するコマンドインタプリタで、試験プログラムのグループ（２３０）内で最初に起動されるリーダー格である。コマンドインタプリタは、試験プログラム終了後も、仮想ターミナル（１３０）と接続が維持される限り、ターゲットマシン上に常駐する。

【００３０】２３２は、ユーザレベルデバッガで、コマンドインタプリタ（２３１）上で起動される。２３３は、試験プログラムで、ユーザレベルデバッガ（２３２）を介して起動される。３００は、通信路で、ＯＳ検証装置（１００）とターゲットマシン（２００）を接続している。媒体としては、例えば、ＲＳ２３２Ｃやイーサネットを用いる。この通信路を用いて、試験プログラムをＯＳ検証装置からターゲットマシンに転送したり、ターゲットマシン上で動作するプログラムの標準入出力をＯＳ検証装置に向けることが可能になっている。

【００３１】図２は、試験プログラムが試験項目別のフォルダによって管理される様子を表わしている。試験項目毎のフォルダ（１ａ）は、階層構造を有しており、同フォルダの階層的に下位のフォルダとして、試験プログラムを生成するためのソースファイルを収めるｓｒｃフォルダ（２）と試験プログラムの実行時にファイルを収めるｂｉｎ（３）フォルダが存在する。ｓｒｃフォルダ（２）の中には、試験プログラムのソースファイル（２１）、ソースファイルから試験プログラムを生成する手続きを記述したファイル（２２）がある。ｂｉｎフォルダ（３）の中には、前記試験プログラムのソースファイル（２１）から生成される試験プログラムのロードモジュール（３１）、試験プログラムの実行に関する定義（５）を収める。

【００３２】試験プログラムの実行に関する定義（５）は、試験プログラムのロードモジュール名に“．ｒｕｎ”の名前で定義することで、ＯＳ検証装置は、試験プログラムのロードモジュール（３１）に対する試験プログラムの実行に関する定義（５）を把握することができる。

【００３３】図３は、試験プログラムのパス情報を記述するファイル（４）とその形式を示している。試験プログラムのパス情報を記述するファイルの中には、試験プログラムのパス（４１）が１行に１つの割合で記述されている。本実施の形態では、試験プログラム実行時にユーザレベルデバッガを介するため、試験プログラムの実行時にソースファイルを必要とする。試験プログラムは、図２に示したように、階層構造を持つフォルダで管理されており、試験プログラムのパス（４１）中、最後が試験プログラムのロードモジュールのファイル名、その前がｂｉｎフォルダ、その前が試験項目名となっている。試験項目名の一例“／ｋｅｒｎｅｌ／ｓｙｓｃａｌｌ／ｒｅａｄ／”を図３に示している。これにより、試験自動実行化部（１２０）は、試験プログラム実行時に、ターゲットマシンでのユーザレベルデバッガの動作形態に応じて必要な試験項目毎のフォルダ（１ａ）の中身を、ターゲットマシンに転送することが可能になっている。

【００３４】或は、この下のｂｉｎフォルダ（３）の中身を転送する場合もある。ｂｉｎフォルダ（３）の中身だけを転送する処理は、ＯＳ検証装置（１００）がターゲットマシン（２００）のクロス開発ホストとなっていて、ＯＳ検証装置（１００）上にあるユーザレベルデバッガ本体がＯＳ検証装置（１００）上で動作し、ユーザレベルデバッガの一部と試験プログラムがターゲットマシン（２００）上で動作する試験環境に対応する。ＯＳ検証装置は、試験プログラムのパス情報を記述するファイル（４）中の試験プログラムのエントリを１つずつ処理する。これにより、ＯＳ検証装置は、ターゲットマシン上で実施中の試験プログラムを把握している。

【００３５】図４は、試験プログラム実行に関する定義（５）とその形式を示している。試験プログラム実行に関する定義（５）は、試験項目毎のフォルダ（１ａ）内に、試験プログラムに対して１つの割合で存在する。原則として、試験プログラム名に対する拡張子（．ｒｕｎ）により、試験自動実行化部（１２０）が対応を把握できる仕掛けになっている。試験プログラム実行に関する定義（５）には、本ＯＳ検証システム上で試験プログラムを実行する際に必要となる、以下の情報がある。

【００３６】５１は、試験プログラム実行時の引数情報である。本実施の形態では、先頭行に記している。５２は、試験プログラム実行時に設定するブレイクポイントに関する情報である。本実施の形態では、２行目以降

に、１つのブレイクポイントに対して１行を用いて記載している。ブレイクポイントに関する情報（５２）は、試験プログラムを構成するファイルをデバッガ上でリスト表示する際にキーワードとなる関数名（５３）と試験プログラムを構成するファイルでの行番号（５４）から成る。尚、試験プログラムにおけるブレイクポイントの設定場所は、原則として、試験プログラムがエラーを検出した場所として、試験実施者が選ぶ。

【００３７】図５は、実施の形態１を実現する、ＯＳ検証装置内の試験自動実行化部の処理の流れを示している。以下に、各ステップでの処理を説明する。

【００３８】ステップ９００では、ＯＳ検証装置（１００）上に仮想ターミナル（１３０）を準備し、仮想ターミナルの通信機能を用いて、ターゲットマシン（２００）上にコマンドインタプリタ（２３１）の起動を要求し、生成されたコマンドインタプリタ（２３１）と接続する。この仮想ターミナル（１３０）上での操作を通して、ターゲットマシン（２００）上で、ユーザレベルデバッガ（２３２）と試験プログラム（２３３）の各種コマンドを起動し、試験対象のＯＳ（２２０）を試験する。

【００３９】ステップ１０００では、試験プログラムのパス情報（４）を読み込む。例えば、試験自動実行化部（１２０）では、試験プログラムのパス情報（４）を記述するファイル名を起動時の引数として渡すか固定のパスに設定しておくことで扱うことが可能になっている。試験自動実行化部（１２０）は、ステップ１０００からステップ１７００迄の１回のループ処理で、１度に１個の試験プログラムを扱う。このため、ステップ１０００では、試験プログラムのパス情報（４）において先頭から順繰りに試験プログラムのエントリを読み込む。

【００４０】ステップ１１００では、ステップ１０００での処理で読み込んだパス情報を元にして、実施する試験があるかないかで処理を変える。実行する試験の記述が存在しないとは、試験プログラムのパス情報（４）ファイルでの最終行迄行き着いたことをいう。実行する試験の記述が存在する場合は、ステップ１２００へ移り、実行する試験の記述が存在しない場合は、試験自動実行化部の処理を終了する。

【００４１】ステップ１２００では、ステップ９００で準備した仮想ターミナル（１３０）に対して、オペレータがキータイプするのと同等の効果を生じさせることを自動化した手段（以下、「自動化手段」ともいう）を用いて実現する。仮想ターミナル（１３０）に対して行われるキータイプ相当の処理で渡る文字列は、仮想ターミナル（１３０）とターゲットマシン（２００）上でのコマンドインタプリタが接続されているので、内容はコマンドインタプリタによって解釈される。ここでの目的は、ターゲットマシン（２００）上でユーザレベルデバッガ（２３２）を介して試験プログラム（２３３）を起動することであるが、これは以下のようにして行う。

【００４２】まず、ターゲットマシン上で試験プログラムを実行するために必要な試験セットをターゲットマシンに転送する。例えば、リモートコピーコマンドの実行をコマンドインタプリタに解釈させ、試験項目毎のフォルダ（１）全体をターゲットマシン上の２次記憶装置（２１０）に転送する。リモートコピーコマンド実行時に必要となる転送すべきフォルダのパスは、試験プログラムのパス情報（４）から、試験プログラム名とｂｉｎフォルダ以下を除いたパスとして得ることができる。具体的には、ＯＳ検証装置内では、“ｒｃｐ　－ｒ　ＯＳ検証装置名：転送すべきフォルダのパス名　．（改行）”といった文字列を、仮想ターミナル（１３０）上に送る。ターゲットマシン上への試験プログラムの転送が完了後、仮想ターミナル（１３０）上へ、ユーザレベルデバッガのパスとそれに続けて試験プログラム名から成る文字列送る。この文字列は、ターゲットマシン上のコマンドインタプリタによって解釈されて、ターゲットマシン上でユーザレベルデバッガを介して試験プログラムが起動される。この時点で、試験プログラムはまだ実行開始していない。

【００４３】ステップ１３００では、試験プログラムの実行に関する定義（５）ファイルを読み込む処理である。この処理で、試験プログラム実行時の引数（５１）と、試験プログラム中に設定するブレイクポイントに関する情報（５２）を得る。本実施の形態では、試験プログラムの実行に関する定義（５）のファイルは、図２に示したように、現在実行中の試験プログラムのパスに拡張子“．ｒｕｎ”を付けたものとして定義して、試験プログラムが存在するフォルダに存在する。

【００４４】ステップ１４００では、ステップ１２００に記載した自動化手段を用いて、ステップ１２００の処理でターゲットマシン（２００）上に起動したユーザレベルデバッガ上で、試験プログラム（２３３）にブレイクポイントを設定する。ここでは、ユーザレベルデバッガに、ステップ１３００で読み出した試験プログラム中に設定するブレイクポイントに関する情報（５２）のうち、５３を用いて試験プログラムを構成するソースファイルを読み込ませ、次に、試験プログラムを構成するファイルでの行番号（５４）にブレイクポイントを設定する。ブレイクポイントを複数設定する必要がある場合、ユーザレベルデバッガ上での本操作を繰り返し実行する。

【００４５】ステップ１５００では、ステップ１２００に記載した自動化手段を用いて、ユーザレベルデバッガで起動した試験プログラムを、ステップ１３００で得た試験プログラム実行時の引数を伴なわせて開始させる。具体的には、仮想ターミナル（１３０）に対して、ターゲットマシン上のユーザレベルデバッガが解釈して試験プログラムを所望の引数を伴なって開始させる文字列、

図4の例では、“runarg1　arg2　arg3（リターン）”を送る。

【0046】ステップ1600では、仮想ターミナル（130）上での自動対話操作（画面キャプチャ）により、試験プログラム（233）の終了を監視する処理である。本処理は一定時間（例えば、1秒）毎に、仮想ターミナル（130）の画面をキャプチャし、仮想ターミナルのサイズに応じた文字列配列に文字データを得る。例えば、80桁×24行の配列に仮想ターミナル（130）上に出力されている文字を取り込む処理を行う。具体的には、仮想ターミナル（130）上で試験プログラムとユーザレベルデバッガの標準出力が表示され、表示内容が上方向にスクロールされる状況において、検査者は、画面上の最下位行（24行目）に表示されるユーザレベルデバッガのプロンプトと、その前に出力されたことで1行上（23行目）に表示されている筈のユーザレベルデバッガによる試験プログラムの状態を示す情報を文字列データとして得て監視する。上記2行分の文字列データを、文字列の比較処理を行うことで、試験の実行状況を把握する。ちなみに、試験プログラムが終了していないうちは、デバッガのプロンプトは表示されないので、画面キャプチャした画面上の24行目に相当する文字列データが、デバッガのプロンプトを表わす文字列になっていない。

【0047】ステップ1700では、ステップ1600での処理に基づき、試験プログラム（233）がブレイクポイントで停止した場合には終了する。試験プログラムがブレイクポイントで停止しなかった場合にはステップ1000に戻る。ブレイクポイントで停止していた場合の状態を示す例としては、ステップ1600の処理で仮想ターミナル（130）を画面キャプチャしたデータにおいて、画面上の24行目に相当する文字列データがユーザレベルデバッガのプロンプトと同じであり、更に23行目に相当する文字列データがブレイク場所を示しており、更に22行目に相当する文字列データがブレイクポイントで停止したことを示している。

【0048】以上の試験自動実行化部の処理により、試験プログラムの実行中にブレイクポイントが検出されなければ、試験プログラムのパス情報（4）に基づいて試験プログラムが順次自動実行される。一方、ブレイクポイントが検出された場合は、ステップ1400の処理で設定したブレイクポイントによって、試験プログラムは停止したままとなる。このようにして、試験プログラムの所望の箇所で試験プログラムの実行を停止させる。

【0049】このように、試験対象とする計算機のオペレーティングシステムを、OSが提供するアプリケーション・プログラミング・インタフェースを用いてOSのサービス内容を確認する試験プログラムによって検証するソフトウェア検証装置において、前記試験プログラムをOS上に実現されたユーザレベルデバッガを介して起動し、試験プログラムがエラーする箇所にブレイクポイントを設定した後に実行開始することで、障害検出後に試験プログラムを停止させ、障害を検出した状態を保持させたままとすることを特徴とする。

【0050】実施の形態2．実施の形態2では、上記実施の形態1のようにして、試験プログラムが設定されたブレイクポイントで停止した場合、試験プログラムが用いている変数の値（引数情報）を取得する障害情報の収集処理機構を実現する場合を説明する。本実施の形態では、試験プログラムが障害検出時の試験対象プログラムの内部状態を得ているが、これは試験対象プログラムが障害を起こす手続きの把握や、試験プログラムの誤りによって試験対象プログラムが障害と判定される問題を回避可能にする。この実施の形態も実施の形態1と同様に、OS検証装置を一例として説明する。

【0051】図6は、実施の形態2を実現するシステムの一例の構成図である。140は、OS検証装置における障害情報収集部で、図9に示す動作を行う。111は、障害情報である。本実施の形態における障害情報は、障害情報収集部（140）によって収集される、ターゲットマシン上で停止中の試験プログラムのデータである。障害情報は、ターゲットマシン上で停止中の試験プログラムの名前と関連付けられて、OS検証装置の2次記憶装置に記録される。図6では、障害情報（111）は、記憶部（110）とは別の記憶装置として表わしているが、同じ記憶装置であってもよい。

【0052】図7は、試験プログラム内部のシンボル情報を表わす試験プログラムシンボル情報のファイル（6）とその形式を示している。シンボル情報は、プログラムがメモリ上に展開されるときにプログラム内部で用いているデータ及び関数の先頭（シンボルと呼ぶ）が割り当てられるアドレスを示す。通常、割り当てられるアドレスは、プログラムのソースファイルをコンパイル後、リンク処理されたときに決まるものである。シンボル情報は、ロードモジュールに含まれている。この実施の形態では、必要に応じて障害情報収集部（140）がシンボル情報を取得する。これについては、後述する（図9、ステップ1722）。試験プログラム内部のシンボル情報を表わすファイル（6）は、例えば、標準UNIX（「UNIX」は、登録商標）においては、試験プログラムのロードモジュールにシンボルテーブルが含まれる状態で、標準nmコマンドの引数に試験プログラムのロードモジュールのパスを指定して実行することで得られる。

【0053】61は、シンボルのアドレスを表わす。62は、シンボルの識別子を表わす。シンボルの識別子は、シンボルがデータであるか関数の先頭であるかを示す。ここでの情報で、データであるシンボルを判別する。63は、シンボル名を表わす。シンボルの種類がデータである場合、変数データの名前である。このシンボ

ル情報から得る変数データは、試験プログラムがブレイクポイントで停止した場合に、その時点での値を障害情報として収集する対象となる。

【００５４】図８は、実施の形態２を実現する、試験自動実行化部（１２０）の処理の流れの一例を表わした図を示している。本処理は、図５に対して、ステップ１７００の処理を変更してステップ１７１０とし、ステップ１７２０の処理を追加している。以下に、これら変更したステップでの処理について説明する。

【００５５】ステップ１７１０では、試験プログラム（２３３）がステップ１４００の処理で設定したブレイクポイントで停止したかどうかで処理を変えている。ブレイクポイントで停止していた場合は、ステップ１７２０の処理に移る。ブレイクポイントで停止せずに終了した場合は、ステップ１０００の処理に戻る。ブレイクポイントで停止していた場合の状態を示す例としては、ステップ１６００の処理で仮想ターミナル（１３０）を画面キャプチャしたデータにおいて、画面上の２４行目に相当する文字列データがユーザレベルデバッガのプロンプトと同じであり、更に２３行目に相当する文字列データがブレイク場所を示しており、更に２２行目に相当する文字列データがブレイクポイントで停止したことを示している。

【００５６】ステップ１７２０では、障害情報収集部（１４０）の処理を行う。ステップ１７２０の処理の詳細は、図９の流れで示している。

【００５７】次に、図９について説明する。ステップ１７２１では、仮想ターミナル（１３０）上での自動対話操作（画面キャプチャ）により、試験プログラム（２３３）の停止行番号を獲得して、障害情報（１１１）を残す処理である。ユーザレベルデバッガ上で設定したブレイクポイントで停止していた場合、例えば、画面キャプチャしたデータにおいて、仮想ターミナル（１３９）上の２４行目にユーザレベルデバッガのプロンプト、２３行目に停止行番号、２２行目にブレイクポイントで停止した旨と停止位置情報（試験プログラムのソースファイルと停止行番号）が示されている。

【００５８】本ステップでは、障害情報（１１１）をファイルに残すが、このファイル名として試験プログラム名に“．ｌｏｇ”の拡張子を付けたものを生成し、上記キャプチャ処理して得たデータのうち２２，２３行目で得た情報を書き込む。この内容の一例を、図１０中に試験プログラムの停止箇所（１２）として示した。

【００５９】ステップ１７２２では、実行中の試験プログラム（２３３）のロードモジュールからシンボルテーブルを得て、試験プログラム（２３３）内部のデータ名を得る処理である。本実施の形態では、記憶部（１１０）上の試験プログラムのロードモジュールにおいて、ｎｍコマンドを用いて、図７に示した形式の試験プログラム内部で用いているシンボルの情報をファイルに得る。次に、このファイルにおいて、シンボルの識別子がデータを表わしているものを抽出して、最終的に試験プログラム内部の変数名を得る。

【００６０】ステップ１７２３では、仮想ターミナル（１３０）上でのステップ１２００記載の自動化手続きと画面キャプチャを行い、試験プログラム（２３３）内部の変数について、試験プログラムがエラーを検出して停止している状態での値を獲得して、障害情報（１１１）として記録する。具体的には、ステップ１７２２の処理で得た試験プログラム内部の変数名において、仮想ターミナルに文字列を送信して、ユーザレベルデバッガ上でデータ値を表示させた後、ユーザレベルデバッガの出力を画面キャプチャして、ステップ１７２１で作成した障害情報を記すファイルに記録する。

【００６１】例えば、ユーザレベルデバッガが、所望変数の表示手続き（改行）、所望変数の値（改行）、プロンプト（改行）と出力したものは、２４行×８０桁の仮想ターミナル上で画面スクロールされると、２２行目に所望変数の表示手続き、２３行目に所望変数の値、２４行目にプロンプトが表示される。画面キャプチャ機能により、これらを、文字列として得て、プログラム処理することが可能になる。ここでのステップで得る障害情報は、図１０において、所望変数の表示手続き（１１３）と、所望変数の値（１１４）である。

【００６２】ステップ1･7２４では、仮想ターミナル（１３０）上でのステップ１２００記載の自動化手続きによって、停止した試験プログラム（２３３）のスタックのバックトレース等試験プログラム実行時のフローを解析するのに役立つ情報表示をユーザレベルデバッガに行わせる。尚、本処理は、使用するユーザレベルデバッガの機能依存である。

【００６３】例えば、バグトレースを行うにあたって、仮想ターミナルに対して“ｂｔ（リターン）”という文字列を送信して、ユーザレベルデバッガに指示を与える。これにより、ユーザレベルデバッガは、停止している試験プログラムのスタックフレームを解析する。この結果、現在の停止地点がどういった経歴で呼ばれてきたかを示す停止しているポイントを含む関数名、その関数を含むソースファイル名が得られる。

【００６４】更に、ユーザレベルデバッガでの結果結果を仮想ターミナル上で行う画面キャプチャによって取得し、ステップ１７２３での処理と同様に、障害情報（１１１）として残す。これらは、図１０中のユーザレベルデバッガへの指示（１１５）とユーザレベルデバッガによる解析結果（１１６）として示した。

【００６５】本実施の形態では、ユーザレベルデバッガから得る試験プログラムの変数値を画面キャプチャ機能によって取得したが、仮想ターミナルにファイルへのログ出力機能がある場合は、ステップ１７２１の処理開始時点でこのロギング機能をオンして２次記憶装置上の障

害情報（１１１）を格納するファイルと関連付け、ステップ１７２４の処理終了時点でロギング機能をオフすることで、ユーザレベルデバッガへの自動対話操作はキータイプのみ行なえば良くなる。

【００６６】このように、このソフトウェア検証装置は、試験プログラムが設定したブレイクポイントで停止した後、試験プログラムが用いている変数名を試験プログラムのロードモジュールが含むシンボルテーブルから得て、試験プログラムを動作させているユーザレベルデバッガを通して停止中の試験プログラムの変数値を取得する障害情報の収集処理を備えたことを特徴とする。

【００６７】実施の形態３．実施の形態３では、試験プログラムが設定されたブレイクポイントで停止した後、試験対象のＯＳが用いている変数の値を取得する障害情報の収集処理機構を実現する。

【００６８】図１１は、実施の形態３を実現するシステムの構成図を表わしている。本実施の形態では、ＯＳ検証装置（１００）上で動作する通信機能を有した仮想ターミナル（１５０）上で、ターゲットマシン（２００）上のＯＳをデバッグすることが可能な構成になっている。

【００６９】１５０は、ＯＳ検証装置上で動作する仮想ターミナルである。本仮想ターミナルは通信機能を用いて、ＯＳ検証装置（１００）上からターゲットマシン上の試験対象のＯＳをデバッグ動作させるために、ターゲットマシン上のカーネルレベルデバッガと接続する。尚、ＯＳ検証装置（１００）上で、本仮想ターミナル（１５０）への入出力は、障害情報収集部（１４０）があたる。

【００７０】２２１は、カーネルレベルデバッガで、試験対象のＯＳ内に実現されている。カーネルレベルデバッガは、ターゲットマシン上の特定の通信ポートに関連付けられており、この通信ポートを用いて入出力することで、試験対象のＯＳをデバッグ動作させることが可能になっている。

【００７１】7は、試験対象のＯＳのロードモジュールである。8は、試験対象のＯＳ内部で用いているシンボル情報を表わすファイル（単に、「試験対象のＯＳ内部で用いているシンボル情報」ともいう）である。内容は、図１２に示す。

【００７２】図１２は、試験対象のＯＳ内部で用いているシンボル情報を表わすファイル（８）とその形式を示している。シンボル情報とは、ＯＳがメモリ上に展開されたときにＯＳ内部で用いているデータ及び関数名（シンボル）が割り当てられるアドレスを示す情報である。通常、アドレスは、プログラムのソースファイルをコンパイル後、リンクしたときに決まるものである。シンボル情報は、ロードモジュールに含まれている。また、シンボル情報は、リンク処理時にファイル等に出力することが可能である。

【００７３】例えば、ＵＮＩＸにおいては、ＯＳのシンボル情報は、ＯＳのロードモジュールにシンボルテーブルが含まれる状態で、ｎｍコマンドの引数にＯＳのロードモジュールのパスを設定して実行することで得られる。また、これは、ＯＳ生成時に作成を要求することができる。本実施の形態では、ＯＳのロードモジュールから得る方法について扱う。

【００７４】８１は、シンボルのアドレスを表わす。８２は、シンボルの識別子を表わす。ここでの情報で、データであるシンボルを判別する。８３は、シンボル名を表わす。シンボルの種類がデータである場合、変数データの名前である。このシンボル情報から得る変数データは、試験プログラムがブレイクポイントで停止した場合に、その時点での値を障害情報として収集する対象となる。

【００７５】図１３は、この実施の形態における障害情報収集部の処理の流れを表わしている。図３は、図９に表わした障害情報収集部の処理の流れに加えて、ステップ１７２５とステップ１７２６を拡張した。以下に、拡張したステップでの処理ついて説明する。ステップ１７２５では、試験対象のＯＳのロードモジュール内部で用いているシンボル情報（８）から、ＯＳ内のデータ名とそれが割り当てられたアドレスを得る処理である。図１２に示した情報において、アドレス（８１）がデータ空間に含まれるシンボル名（８３）を以下の方法で得る。記憶部（１１０）上の試験対象のＯＳのロードモジュール（７）において、ｎｍコマンドを用いて、図１２に示した形式の試験対象のＯＳ内部で用いているシンボルの情報をファイルに得る。

【００７６】ステップ１７２６では、仮想ターミナル（１５０）上で行うステップ１２００記載の自動化手続きにより、試験対象のＯＳ（２２０）が使用しているデータについて、試験プログラム（２３３）がエラーを検出して停止している状態での値を表示させ、出力結果を画面キャプチャによって獲得して、障害情報（１１１）として記録する。

【００７７】具体的には、ステップ１７２５の処理で得た試験対象のＯＳ内部のシンボルの情報をステップ１７２１で作成した障害情報（１１１）を記すファイルに追記した後（図１４中の８）、続けて、仮想ターミナル（１５０）へのキータイプによりカーネルレベルデバッガ上でカーネルのデータ空間をダンプする操作を行なった後、画面キャプチャして、キャプチャしてデータを障害情報（１１１）を記すファイルに追記する（図１４中のカーネルレベルデバッガへの要求（１７）とカーネルレベルデバッガによる出力（１８））。ここで得られる情報により、特定アドレスのメモリの内容が、実際にＯＳのどの変数に割り当てられているかを知ることが可能になる。強いては、試験プログラムが障害検出時のＯＳ内部の変数値を知ることができる。

【００７８】本実施の形態では、カーネルレベルデバッガから得る試験対象のＯＳ内部のデータ値を画面キャプチャ機能によって取得したが、仮想ターミナルにファイルへのログ出力機能がある場合は、ステップ１７２５の処理開始時点でこのロギング機能をオンして２次記憶装置上の障害情報（１１１）を格納するファイルと関連付け、ステップ１７２６の処理終了時点でロギング機能をオフすることで、仮想ターミナル（１５０）への自動対話操作はキータイプのみ行なえば良くなる。

【００７９】このように、このソフトウェア検証装置は、試験プログラムが設定したブレイクポイントで停止した場合、試験対象のＯＳが用いている変数名をＯＳのロードモジュールが含むシンボルテーブルから得て、試験プログラムによって試験された状況下でのＯＳ内部の変数が保持する値をカーネルレベルデバッガを通して取得する障害情報の収集処理を備えたことを特徴とする。

【００８０】実施の形態４．実施の形態４では、試験プログラムが設定されたブレイクポイントで停止した後、試験プログラムが使用するＯＳが管理する資源の状態を取得する、障害情報の収集処理機構を実現する。

【００８１】図１５は、実施の形態４を実現するシステムの構成図を表わしている。２４１は、ターゲットマシン（２００）上で動作する障害情報を収集する際に利用する障害用コマンドインタプリタである。本障害用コマンドインタプリタは、試験開始時、ＯＳ検証装置（１００）からの要求によって生成され、試験実施中ターゲットマシン上に常駐する。１６０は、ＯＳ検証装置上で動作する仮想ターミナルで、通信機能によって、ターゲットマシン上で動作する障害用コマンドインタプリタ（２４１）から起動されるプログラムのグループ（２４０）と接続されている。ＯＳ検証装置（１００）上で、本仮想ターミナルへの入出力は、障害情報収集部（１４０）があたる。２４０は、ターゲットマシン（２００）上で試験対象のＯＳ（２２０）上で動作する障害用コマンドインタプリタ（２４１）が起動するプログラムのグループである。障害情報収集部（１４０）が仮想ターミナル（１６０）を介して行う障害用コマンドインタプリタ（２４１）に対する要求によって、ターゲットマシン（２００）上で各種プログラム（２４２）が生成され、これらのプログラムの標準出力が仮想ターミナル（１６０）上で確認可能となっている。１４０は障害情報収集部である。障害情報収集部（１４０）の処理は、本実施例において拡張される。詳細は、図１８に示す。

【００８２】図１６は、試験プログラムの実行に関する定義のファイル（５）とその形式の一例を表わしている。図４に表わした定義のファイル（５）に比べて拡張される要素は、以下の通りである。５５は、ターゲットマシン上のコマンドインタプリタと接続された仮想ターミナル上で実施する自動対話手続き（オペレータのキータイプ相当の処理の内容）を示している。本実施例では、“ＣＭＤ：”という識別子に続けて記された文字列（図１６中のｐｓ　－Ａｌ等）とリターンコードが、仮想ターミナルに送られる。

【００８３】図１７は、この実施の形態の試験自動実行化部（１２０）の処理を示している。図１７は、図５に表わした試験自動実行化部（１２０）の処理にステップ９００の処理を拡張した。以下に、拡張したステップについて説明する。ステップ９１０では、ＯＳ検証装置（１００）上に仮想ターミナル（１６０）を準備し、仮想ターミナルの通信機能を用いて、ターゲットマシン（２００）上にコマンドプロンプトを生成する。ＯＳ検証装置では、この仮想ターミナル（１６０）上での操作を通して、ターゲットマシン（２００）上に各種プログラム（２４２）の各種コマンドを起動し、その結果を得ることが可能になっている。

【００８４】図１８は、この実施の形態の障害情報収集部（１４０）の処理を表わしている。図１８は、図１３に表わした障害情報収集部（１４０）の処理にステップ１７２７とステップ１７２８を拡張した。以下に、拡張したステップでの処理について説明する。

【００８５】ステップ１７２７では、ターゲットマシン上の２次記憶装置（２１０）の情報等、試験プログラムの実行に影響がある環境に関する情報を得るために、ターゲットマシン上の障害用コマンドインタプリタ（２４１）上で行う手続きを得る。本実施の形態では、ターゲットマシン上の障害用コマンドインタプリタ（２４１）上で行う手続きは、図１６に示す試験プログラムの実行に関する定義（５）において、識別子“ＣＭＤ：”で始まる文字列で記されている。

【００８６】ステップ１７２８では、仮想ターミナル（１６０）上で行うステップ１２００に記載した自動化手続きによって、ステップ１７２７の処理で得た試験プログラムが使用するＯＳが管理する資源の状態を得る操作を表わす文字列を、障害用コマンドインタプリタ（２４１）に送る。コマンドインタプリタは、この文字列を解釈して、指示された操作を行う。

【００８７】これにより得られる結果出力は、仮想ターミナル（１６０）を画面キャプチャして、仮想ターミナルサイズに応じた文字列データ（２４行×８０桁）に変換し、障害情報（１１１）としてファイルに記録する。ここでのステップで得られる障害情報は、図１９中に、コマンドインタプリタへの指示（１１７）、ＯＳが管理する資源の状態（１１８）として示した。

【００８８】本実施の形態では、障害用コマンドインタプリタ（２４１）上で行なった操作によって得る試験環境に関する情報を仮想ターミナル（１６０）において行なう画面キャプチャによって取得したが、仮想ターミナルにファイルへのログ出力機能がある場合は、ステップ１７２８の処理開始時点でこのロギング機能をオンしてログファイルを２次記憶装置上の障害情報（１１１）を

格納するファイルと関連付け、ステップ１７２８の処理終了時点でロギング機能をオフすることで、仮想ターミナル（１６０）への自動対話操作はキータイプのみ行なえば良くなる。

【００８９】このように、このソフトウェア検証装置は、試験プログラムが設定したブレイクポイントで停止した後、試験プログラムが使用する資源で、その状態によりＯＳの処理が変わることで試験プログラムの実行結果に影響するような試験プログラムの実行環境に関する情報を得るための操作を、試験対象のＯＳ上で動作するコマンドインタプリタ上で行い、この結果を得る障害情報の収集処理を備えたことを特徴とする。

【００９０】実施の形態５．実施の形態５では、請求項５に関連して、試験プログラムにハングアップ判定を可能とするタイムアウト時間の設定と試験プログラムハングアップ後に行う対処の定義を行うことを可能とし、タイムアウト時間経過後に試験プログラムが終了していなかった場合は試験プログラムがハングアップを起こしたと判定して、試験プログラム停止、障害情報収集、試験プログラム終了、ハングアップ時の対処といった一連の処理を自動化する機構を実現する。

【００９１】図２０は、実施の形態５を実現するためにメンバが拡張された試験プログラムの実行に関する定義（５）を表わしている。以下に、拡張したメンバについて説明する。５６は、タイムアウト時間である。ここに示す時間が経過しても試験プログラムが終了しない場合、試験プログラムがハングアップを起こしたと判断する。本実施の形態では、識別子“ＴＯＵＴ：”後に数字で表記された数（図２０中１０と例示）を秒数として扱っている。

【００９２】５７は、試験プログラムがハングアップしたと判定した後の対処である。本実施の形態では、仮想ターミナル（１６０）上に、ステップ１２００に記載した自動化手続きによって、識別子“ＨＵＰ：”後に表記された文字列とリターンコードを送信する。ここには、試験プログラムが使用するＯＳが管理する資源を解放する処理を登録する（図２０の例では、一時ファイル（＊．ｔｍｐ）の消去をｒｍコマンドにより行う）。これにより、試験プログラムが最後迄終了しないことで、ＯＳが管理する資源を解放しなことで生じる問題を解決する。

【００９３】図２１は、試験自動実行化部（１２０）の処理の流れを表わしている。図２１は、図１７に表わした試験自動実行化部（１２０）の処理にステップ１５５０、ステップ１７３０、ステップ１７４０の処理を拡張し、ステップ１６００の処理をステップ１６０１に変更した。

【００９４】ステップ１５５０では、現時刻を試験プログラム開始時間として取得すると共に、ステップ１３００の処理で獲得した試験プログラムのタイムアウト時間後にタイムアウトが発生する様にアラームの設定を行なっている。

【００９５】ステップ１６０１では、試験プログラムが終了していなくても、ステップ１５５０で設定したアラームが発生した（タイムアウトが生じた）場合、ステップ１７１０の処理へ移る。

【００９６】ステップ１７３０では、タイムアウトが起きたかどうかで処理を変える。タイムアウトが起きていた場合はステップ１７４０の処理へ、起きていない場合はステップ１７００の処理へ進む。

【００９７】ステップ１７４０では、タイムアウトが起きていた場合の処理であるが、詳細は図２２に示す。図２２は、実施の形態５で試験自動実行化部に拡張された、試験プログラムタイムアウト後の処理を表わしている。以下に、拡張された処理の内容を説明する。

【００９８】ステップ１７４１では、試験プログラムを停止させる処理である。例えば、試験対象のＯＳがＵＮＩＸの場合では、ユーザレベルデバッガ上で実施中の試験プログラムにＳＩＧＩＮＴシグナルを送って、試験プログラムを停止させる。例えば、仮想ターミナル（１３０）上で、割り込みコード（ｃｔｒｌ－ｃ）のコードをタイプする。

【００９９】ステップ１７２０では、障害情報収集部（１６２０）が処理を行う。処理の内容は、図１８に示してきた。

【０１００】ステップ１７４２では、仮想ターミナル（１３０）上でのステップ１２００に記載した自動化手続きにより、ユーザレベルデバッガ（２３２）に、試験プログラム（２３３）を終了させる。例えば、仮想ターミナル（１３０）に、“ｑｕｉｔ（リターン）”という文字列を送信する。

【０１０１】ステップ１７４３では、試験プログラムハングアップ後の対処（５７）方法を、試験プログラムの実行に関する定義（５）から読み込む。図２０に示したように、本実施の形態で、試験プログラムの実行に関する定義（５）は、試験プログラムハングアップ後の対処（５７）が追加されている。図２０の例では、“ＨＵＰ：”という文字列に続く、“ｒｍ　＊．ｔｍｐ”という文字列を読み込む。

【０１０２】ステップ１７４４では、ステップ１７４３の処理によって取得した文字列とリターンコードを仮想ターミナル（１６０）上にステップ１２００に記載した自動化手続きによって送信すると共に、試験プログラムがハングアップを起こして実行を中断した旨、障害情報（１１１）に記録する。

【０１０３】このように、このソフトウェア検証装置は、試験プログラムのタイムアウト時間とハングアップ検出後の対処を定義可能としたことで、試験プログラムが試験プログラムのタイムアウト時間を経過しても終了或いは停止しない場合に、試験プログラムがハングアッ

プを起こしたと判断し、ユーザレベルデバッガ上で試験プログラムにシグナルを送信して停止させた後、停止位置を取得すると共に、障害情報の収集を行い、更にユーザレベルデバッガ上で試験プログラムを終了させ、前記ハングアップ検出後の対処を実行する試験の自動化方法を備えたことを特徴とする。

【0104】実施の形態6．実施の形態6では、試験プログラムのハングアップによって試験の自動実行が停止することを防ぎ、且つハングアップを起こした要因を解析するためにデータを自動収集する機能において、効率良くタイムアウト判定時間を設定する機構を実現する。

【0105】図23は、試験自動実行化部（120）の処理の流れを表わしている。実施の形態6を実現するために、図21へステップ1350とステップ1750の処理を拡張した。以下に、拡張したステップでの処理を説明する。

【0106】ステップ1350では、試験プログラムの実行に関する定義ファイル（5）を書き込み可能モードで再オープンする。実施の形態6では、実施の形態5で拡張した試験プログラムの実行に関する定義ファイル（5）（図20）におけるタイムアウト時間（55）を実際に試験プログラムの実行に要した時間を元にした値に書き換えて、次回の試験実施時のハングアップ判定時間を最適化する。

【0107】ステップ1750では、現時刻を試験プログラム終了時刻として取得すると共に、ステップ1550の処理で取得した試験プログラム開始時刻からの経過時間＋αを試験プログラムの実行に関する定義ファイル（5）におけるタイムアウト時間（55）として書き出す。＋αは、ハングアップ判定に余裕をみる時間で、例えば、ステップ1750で算出した試験プログラム開始時刻からの経過時間でも良い。

【0108】このように、ソフトウェアの試験を自動化するソフトウェア検証装置及び方法において、試験プログラムのタイムアウト時間を試験プログラム実施時の実績値を元にして規定する試験プログラムのタイムアウト時間決定する。

【0109】実施の形態7．実施の形態7では、試験プログラムの異常停止によって試験の自動実行が停止することを防ぎ、且つ異常停止を起こした要因を解析するためにデータを自動収集することを可能にする機構を実現する。

【0110】図24は、実施の形態7を実現するために図20を拡張した試験プログラムの実行に関する定義（5）である。以下に、拡張した部分について説明する。拡張は、試験プログラムが想定するシグナルを受けて停止した場合、ユーザレベルデバッガ上で継続動作の処理を行うためのものである。58は、試験プログラムが受信を想定するシグナルに関する情報である。この形式は、シグナル種類：試験プログラムのファイル名：想定するシグナルを受ける行番号（開始）：想定するシグナルを受ける行番号（終了）からなる。

【0111】本実施の形態では、試験対象のOSがUNIXでシグナルが32種類定義されている状況において、シグナル種類の表記は、実行中の試験プログラムがシグナルを受けて停止する際に、ユーザレベルデバッガが表示する形式と同じにしている。

【0112】具体的には、図24中の試験プログラムが想定するシグナル（58）で、シグナル種類はSIGUSR1となっているが、これは、ユーザレベルデバッガ上で得られる表記と同じである。図24に示した試験プログラムが受信を想定するシグナル（58）に関する情報の定義では、SIGUSR1シグナルがtest_src1．cファイルの1行目から100行目以内に受ける可能性があるとしている。

【0113】図25は、試験自動実行化部（120）の処理を表わしている。実施の形態7を実現するために、図23にステップ1760、ステップ1761、ステップ1765の処理を拡張した。以下に、拡張したステップでの処理について説明する。

【0114】ステップ1760では、試験プログラムが異常停止した場合に、ステップ1761へ進み、それ以外はステップ1765へ進む。ユーザレベルデバッガ上での試験プログラムの異常停止は、試験プログラムがシグナルを受信したことによるが、このシグナルは、試験プログラムの記述誤りによって不正処理を行なったためOSから送りつけられることや、OSの障害によって送りつけられる。一方、ユーザレベルデバッガ上で試験プログラムが想定できる停止は、ブレイクポイントを実行したことや、プログラム処理で想定できるシグナルを受けたことによる。

【0115】このことから、試験プログラムが異常停止したことの判断は、試験プログラムが停止している状態を仮想ターミナル（130）を画面キャプチャしてターミナルサイズ（例えば、24行×80桁）の文字列配列に得た後、文字列として得た情報の比較処理によって、停止要因（どのシグナルを受信して、どのファイルの何行目の処理を実行時に停止したか）を把握し、停止要因がブレイクポイントでなく試験プログラムが想定するシグナル（58）でもないことを判定する。

【0116】ただし、シグナルを受けて停止している場所が試験プログラムのソースファイルに含まれていないライブラリ関数内である可能性もあるので、ステップ1724に記したようなバックトレース操作を行なって、試験プログラムでの最終的な呼び出し場所にて判定する。

【0117】具体的には、バックトレース情報を仮想ターミナル画面をキャプチャしてターミナルサイズ（例えば、24行×80桁）の文字列配列に得た後、スタックフレームの解析結果である、試験プログラムが停止して

いる場所を呼んだファイル名とそのファイルでの行番号が、試験プログラムが想定するシグナル（５８）内の範囲に含まれていて、且つ前記停止要因で受信したシグナルと試験プログラムが想定するシグナルが一致している場合、異常停止ではないと判定する。

【０１１８】ステップ１７６５では、ステップ１７６０の判定処理で、試験プログラムが正常処理でシグナルを受けて停止していた場合、ステップ１７６６へ進む。それ以外で終了した場合は、ステップ１７５０へ進む。

【０１１９】ステップ１７６６では、ユーザレベルデバッガ上で試験プログラムを継続動作させる。具体的には、仮想ターミナル（１３０）に対して、ユーザレベルデバッガ上で試験プログラムを継続実行させる自動対話操作を行う。この後、ステップ１６０１の処理へ戻る。

【０１２０】ステップ１７６１は、試験プログラムが異常停止した場合の処理であるが、詳細は図２６に示す。

【０１２１】次に、図２６に示した実施の形態７で試験自動実行化部（１２０）を拡張した、試験プログラムが異常停止した後の処理について説明する。以下に、本実施の形態で拡張したステップでの処理について説明する。ステップ１７２０では、障害情報収集部（１４０）が処理を行う。内容は、図１８に示してきた。

【０１２２】ステップ１７４２では、仮想ターミナル（１３０）でステップ１２００に記載した自動化手続きを行い、ユーザレベルデバッガ（２３２）上で試験プログラム（２３３）を終了させる。例えば、仮想ターミナル（１３０）に対して、“ｑｕｉｔ（リターン）”という文字列を送信する。

【０１２３】ステップ１７４３では、本実施の形態でも実施の形態５と同様に、試験プログラムは最後まで実施されていないため、試験プログラムが使用したＯＳが管理する資源を解放する目的で、試験プログラムの実行に関する定義（５）から、試験プログラムがハングアップした後の対処（５７）を読み込む。

【０１２４】ステップ１７６２では、仮想ターミナル（１６０）に対してステップ１２００に記載した自動化手続きによってステップ１７４３の処理で取得した文字列を送信し、障害用コマンドインタプリタ（２４１）上で試験プログラム異常停止後の処理を実施する。加えて、試験プログラムが異常停止した旨、障害情報（１１１）に記録する。

【０１２５】このように、試験プログラム異常停止後の対処を定義可能としたことで、試験の検証方法において、試験プログラムが異常停止した後、停止位置を取得すると共に障害情報収集部によって障害情報の収集を行い、ユーザレベルデバッガ上で試験プログラムを終了させ、前記試験プログラム異常停止後の対処を実行する試験を自動化することを特徴とする。

【０１２６】実施の形態８．実施の形態８では、試験対象のＯＳの重大障害によりシステムダウンを生じて試験が失敗した後のリカバリを可能とする、試験自動実行の機構を実現する。

【０１２７】図２７は、実施の形態８を実現するために、図１５にターゲットシステムをリセットする機能を拡張したＯＳ検証装置（１００）とターゲットマシン（２００）を表わしている。以下に、拡張部分について説明する。

【０１２８】２５０は、ハードウェアのリセット回路で、外部（接点入力ポート）からの信号入力によってターゲットマシンをリセットする機能を有している。１７０は、出力信号制御ソフトウェアで、接点出力ポートを使用して、ターゲットマシンをリセットするための信号を出力する。尚、１７０と２５０は、低レベルにおいて、電気的に結線されている。

【０１２９】図２８は、実施の形態８を実現するために図２４を拡張した試験プログラムの実行に関する定義（５）である。以下に、拡張した部分について説明する。

【０１３０】５９は、リセット猶予時間である。試験プログラムがハングアップしたと判定後、試験プログラムを停止に導く際に待つ時間で、ここでの時間が経過してもハングアップ状態が続いた場合、ターゲットマシンのハードウェアリセット処理を行う。本実施の形態では、識別子“ＲＳＴ：”後の数字で表記された数を秒数として扱っている。

【０１３１】図２９は、障害情報収集部（１４０）の処理で、実施の形態８を実現するために、図２２に示した障害情報収集部（１４０）の処理を拡張している。以下に、拡張したステップにおける処理について説明する。

【０１３２】ステップ１７７０は、ステップ１７４１の処理で送ったシグナルによって試験プログラムが停止したかどうかで処理を変える。停止した場合はステップ１７２０の処理へ移り、停止しない場合は拡張したステップ１７７１の処理へ移る。

【０１３３】ステップ１７７１では、ハングアップを起こしたと判定されている試験プログラムにおけるここでの処理が、１度目の場合と２度目の場合で処理を変えている。１度目の場合は、ステップ１７７２からステップ１７７５の処理へ移る。２度目の場合、途中にターゲットマシンをリセットする処理を含む、ステップ１７８０からステップ９００の処理へ移る。以下に、ステップ１７７１が１度目であった場合の処理について説明する。

【０１３４】ステップ１７７２では、カーネルレベルデバッガ（２２１）を呼び出す。具体的には、仮想ターミナル（１５０）に対して、オペレータがカーネルレベルデバッガを呼び出す際に行う特定のキータイプと同等の効果を生じるデータ送信処理を行う。

【０１３５】ステップ１７７３では、ステップ１７７２の処理でカーネルレベルデバッガ（２２１）が呼び出せたかどうかで処理を変えている。この判断は、以下のよ

うにして行う。仮想ターミナル（１５０）を画面キャプチャし、仮想ターミナルのサイズに応じた文字列配列に文字データを得る。例えば、８０桁×２４行の配列に仮想ターミナル（１３０）上に出力されている文字を取り込む処理となる。仮想ターミナル（１３０）上でカーネルレベルデバッガの出力が表示され、表示内容が上方向にスクロールされる状況において、画面上の最下位行（２４行目）に表示されるカーネルレベルデバッガのプロンプトが得られたかどうかで、カーネルレベルデバッガを呼び出せたかどうかを判断できる。

【０１３６】カーネルレベルデバッガ（２２１）を呼び出せた場合はステップ１７７４の処理へ移り、カーネルレベルデバッガを呼び出せなかった場合は試験対象のＯＳがハングアップを起こしていると判断してステップ１７８０の処理へ移る。

【０１３７】ステップ１７７４では、カーネルレベルデバッガ（２２１）を終了し、ＯＳの動作を継続させる。具体的には、仮想ターミナル（１５０）に対して、オペレータがカーネルレベルデバッガから抜ける際に行う特定のキータイプと同等の効果を生じる、データ送信処理を行う。

【０１３８】ステップ１７７５では、試験プログラムの実行に関する定義（５）でのリセット猶予時間分の待ち合せを行う。その後、ステップ１６７０の処理へ戻る。次回、ステップ１６７１を通過時は、２度目と判断される。

【０１３９】以下に、ステップ１７７１が２度目であった場合の処理、即ち、試験プログラムが停止しない状態が確実となった後のステップ１６８０からステップ１６８２の処理について説明する。

【０１４０】ステップ１７８０では、出力信号制御ソフトウェア（１７０）を操作して、ターゲットマシンのハードウェアリセット回路（２５０）にリセット信号を送り、ターゲットマシンをリセットさせる。

【０１４１】ステップ１７８１では、試験プログラムが正常終了しないため、ターゲットマシンをリセットさせた旨を障害記録（１１１）に残す。

【０１４２】ステップ１７８２では、ターゲットマシンが起動してくるのを待つ。

【０１４３】上記で説明した障害情報収集部の処理は、実施の形態７で試験プログラムがタイムアウトを生じた後の試験プログラムタイムアウト後の処理（ステップ１７４０）から呼び出させる状況において、ターゲットマシンそのものがハングアップを起こしたことでターゲットマシンを再起動させても、試験プログラムのパス（４）に示されている次の試験を継続して実行させることが可能である。

【０１４４】このように、ソフトウェアの試験の自動化方法において、試験対象のＯＳがハングアップやクラッシュした場合に試験対象のＯＳが動作するターゲットマシンにハードウェアリセット信号を送出し、ターゲットマシンが再起動した後に障害を起こした試験の次からの試験を継続実行する試験の自動化方法であることを特徴とする。

【０１４５】実施の形態９．上記実施の形態１から８では、ソフトウェア検証装置としてＯＳ検証装置を一例として説明した。しかし、上記ソフトウェア検証装置は、ＯＳ以外のソフトウェアとしてアプリケーション・プログラムを検証する場合も含まれることは言うまでもなく、上記以外でも、計算機上で動作するソフトウェアプログラムであって、試験プログラムによって起動できるもを含む。

【０１４６】実施の形態１０．上記実施の形態１から８では、ソフトウェア検証装置とターゲットマシン（２００）とが別の計算機上で実現されている場合を説明したが、図３０に一例として示すように、ターゲットマシン（２００）上に、ソフトウェア検証装置に備えられた機能、例えば、ソフトウェアプログラム群（１）や試験プログラムのパス情報（４）を設置してもよい。

【０１４７】

【発明の効果】以上のように、この発明に係るソフトウェア検証装置又は方法によれば、試験プログラム上に設定したブレイクポイントを通過した場合に試験プログラムが停止したままとなるので、ブレイクポイントを障害を検出した場所に設定することで、障害が生じている状態を保持させたままとすることができる。

【０１４８】また、この発明によれば、試験プログラムが障害を検出した後に試験プログラムが停止したままの状態で、試験プログラムの内部状態を得ることができる。これにより、再現が困難な障害について、障害検出時に障害原因を特定するための情報を得て、試験対象プログラムが障害を起こす手続きの把握や、試験プログラムの誤りによって試験対象プログラムが障害と判定される問題を回避可能にするということができる。

【０１４９】また、この発明によれば、試験プログラムがエラーを検出した後に試験プログラムが停止したままの状態で、試験プログラムが使用したＯＳ内部の資源に関する変数値等のＯＳの内部状態を得ることができる。このため、再現が困難な障害について、障害検出時に障害原因を解析するための情報を得ることができる。

【０１５０】また、この発明によれば、試験プログラムがエラーを検出した後に試験プログラムが停止したままの状態において、試験プログラムが使用するＯＳが管理する資源の状態を得ることができるという効果がある。

【０１５１】また、この発明によれば、試験プログラムがハングアップした時、試験プログラムを停止させ、ハングアップの要因を解析するためのデータを自動収集し、試験プログラムを終了させ、試験プログラムが利用した資源を解放する。これにより、試験プログラムがハングアップした場合でも、要因解析に必要なデータを取

得でき、試験の自動実行を継続させることができるという効果がある。

【0152】また、この発明によれば、試験プログラムのハングアップによって試験の自動実行が停止することを防ぎ、且つハングアップを起こした要因を解析するためにデータを自動収集する機能において、効率良くタイムアウト判定時間を設定することが可能になるという効果がある。

【0153】また、この発明によれば、試験プログラムの異常停止によって試験の自動実行が停止することを防ぎ、且つ異常停止を起こした要因を解析するためにデータを自動収集することを可能にするという効果がある。

【0154】また、この発明によれば、試験プログラムの実行によって試験対象のＯＳが重大障害を起こし、システムダウンしたことで試験が失敗する状況において、試験対象のＯＳが動作するターゲットマシンをハードウェア的にリセットし、ターゲットマシンが再起動した後に障害を起こした試験の次の試験からを継続実行する試験自動実行を可能にするという効果がある。

【図面の簡単な説明】

【図１】　実施の形態１を実現するシステムの一例を表わした図。

【図２】　試験プログラムを管理する試験項目別のフォルダの一例を表わした図。

【図３】　試験プログラムのパスの一例を表わした図。

【図４】　試験プログラムの実行に関する定義の一例をを表わした図。

【図５】　実施の形態１を実現する試験自動実行化部の処理の流れの一例を表わした図。

【図６】　実施の形態２を実現するシステムの一例を表わした図。

【図７】　試験プログラム内部で用いているシンボルの情報の一例を表わした図。

【図８】　実施の形態２を実現する試験自動実行化部の処理の流れの一例を表わした図。

【図９】　実施の形態２における障害情報収集部の処理の一例を表わした図。

【図１０】　実施の形態２における障害情報の一例を表わした図。

【図１１】　実施の形態３を実現するシステムの一例を表わした図。

【図１２】　試験対象のＯＳ内部で用いているシンボルの一例を表わした図。

【図１３】　実施の形態３における障害情報収集部の処理の一例を表わした図。

【図１４】　実施の形態３における障害情報の一例を表わした図。

【図１５】　実施の形態４を実現するシステムの一例を表わした図。

【図１６】　実施の形態４で拡張される試験プログラムの実行に関する定義の一例を表わした図。

【図１７】　実施の形態４で拡張される試験自動実行化部の処理の一例を表わした図。

【図１８】　実施の形態４を実現する障害情報収集部の処理の一例を表わした図。

【図１９】　実施の形態４における障害情報の一例を表わした図。

【図２０】　実施の形態５を実現する試験プログラムの実行に関する定義の一例を表わした図。

【図２１】　実施の形態５を実現する試験自動実行化部の処理の一例を表わした図。

【図２２】　試験自動実行化部における試験プログラムタイムアウト後の処理の一例を表わした図。

【図２３】　実施の形態６を実現する試験自動実行化部の処理の一例を表わした図。

【図２４】　実施の形態７を実現する試験プログラムの実行に関する定義の一例を表わした図。

【図２５】　実施の形態７を実現する試験自動実行化部の処理の一例を表わした図。

【図２６】　実施の形態７を実現する試験プログラム異常停止後の処理の一例を表わした図。

【図２７】　実施の形態８を実現するＯＳ検証装置の一例を表わした図。

【図２８】　実施の形態８を実現する試験プログラムの実行に関する定義の一例を表わした図。

【図２９】　実施の形態８を実現する障害情報収集部の処理の一例を表わした図。

【図３０】　実施の形態１０を実現するシステムの一例を表わした図。

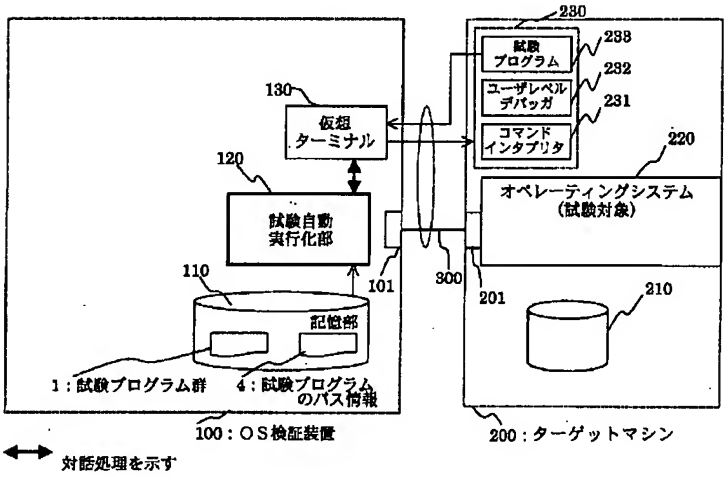
【図３１】　従来システムの構成の一例を示す図。

【符号の説明】

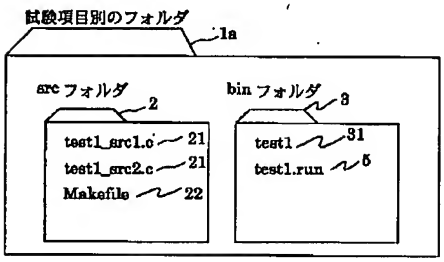
１　試験プログラム群、１ａ　試験項目毎のフォルダ、２　ｓｒｃフォルダ、３　ｂｉｎ、４　試験プログラムのパス情報、５　試験プログラムの実行に関する定義、６　シンボル情報を表わすファイル、７　試験対象のＯＳのロードモジュール、８　試験対象のＯＳ内部で用いているシンボル情報、１０　プログラムテスト装置、１０ａ　テストスクリプト管理手段、１０ｂ　テスト自動化制御手段、１０ｃ　テストスクリプト実行手段、１０ｄ　テスト合否判定手段、１０ｅプログラム障害特定手段、１０ｆ　テストスクリプト入力手段、１０ｇ　テスト結果出力手段、１０ｈ　モード切変え／実行手段、１０ｉ　デバックコマンド入力手段、２０　ターゲットシステム、２１　試験プログラムのソースファイル、２２　試験プログラムを生成する手続きを記述したファイル、３０　検査者、３１　試験プログラムのロードモジュール、４１　試験プログラムのパス、５１引数情報、５２　ブレイクポイントに関する情報、５３　関数名、５４　行番号、５６　タイムアウト時間、５７　ハングアップ時対処、５９　リセット猶予時間、６１　シンボ

ルのアドレス、62　シンボル識別子、63　シンボル名、100　OS検証装置、101　インタフェース、110　記憶部（試験プログラム記憶部）、111　障害情報、120　試験自動実行化部、130　仮想ターミナル、140　障害情報収集部、150　仮想ターミナル、160　仮想ターミナル、200　ターゲットマシン、201　インタフェース、210　2次記憶装置、220　試験対象のOS、221　カーネルレベルデバッガ、230試験プログラムのグループ、231　コマンドインタプリタ、232　ユーザレベルデバッガ、233　試験プログラム、240　プログラムのグループ、241　コマンドインタプリタ、242　各種プログラム、300　通信路。

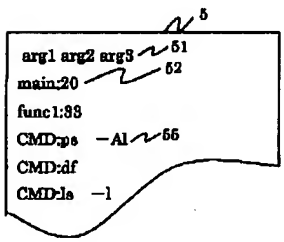
【図1】



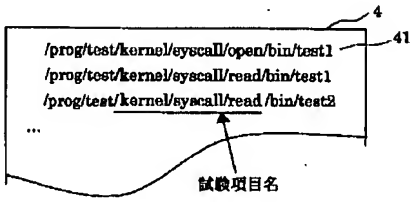
【図2】



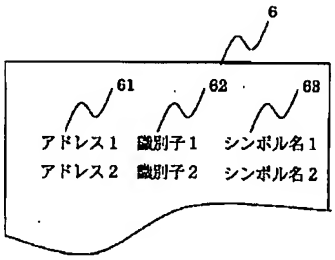
【図16】



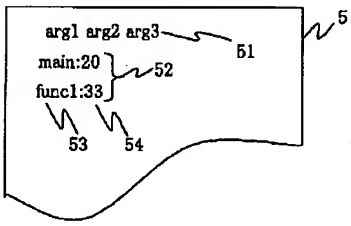
【図3】



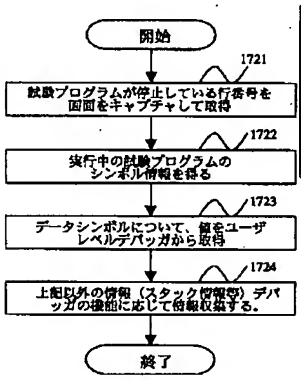
【図7】



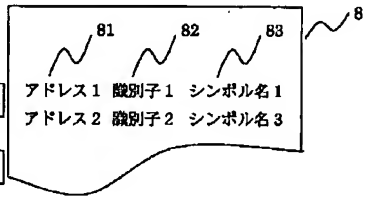
【図4】



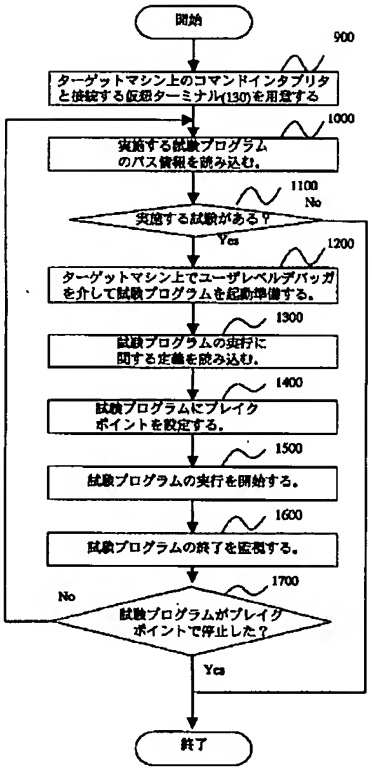
【図9】



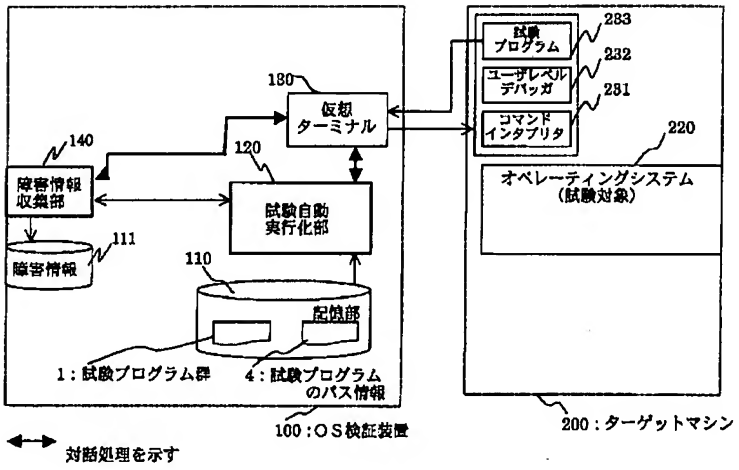
【図12】



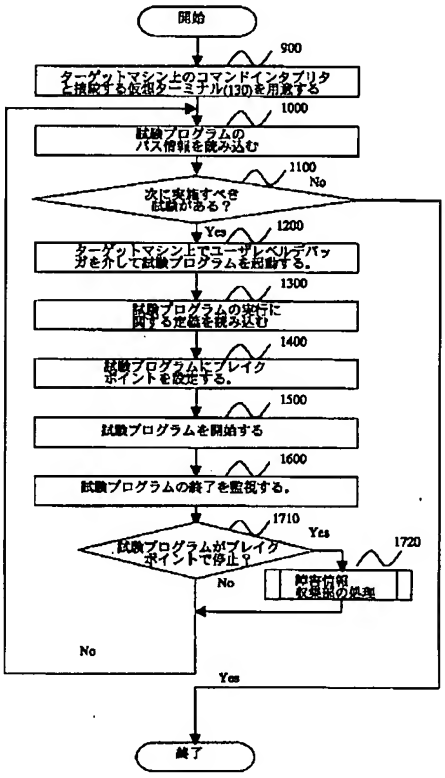
【図5】



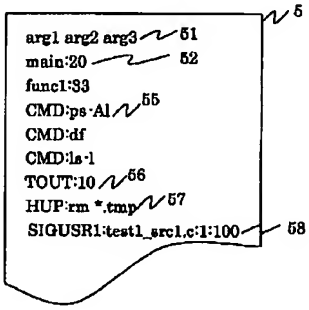
【図6】



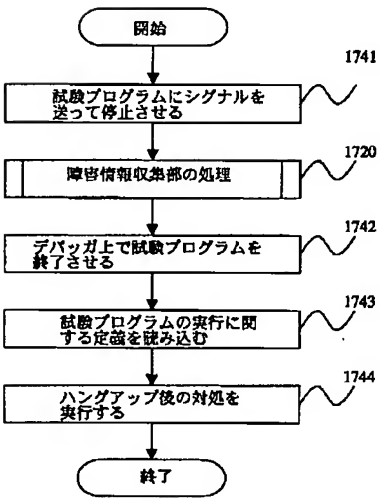
【図8】



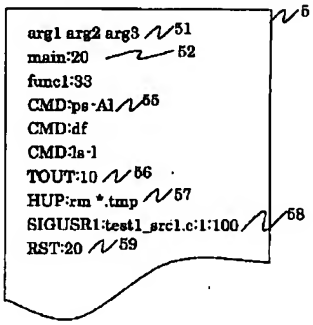
【図24】



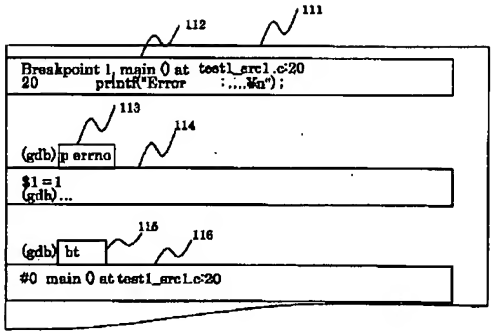
【図22】



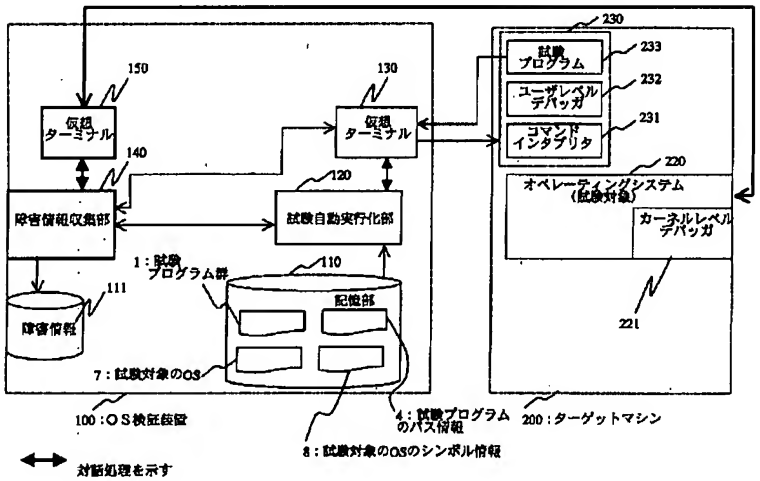
【図28】



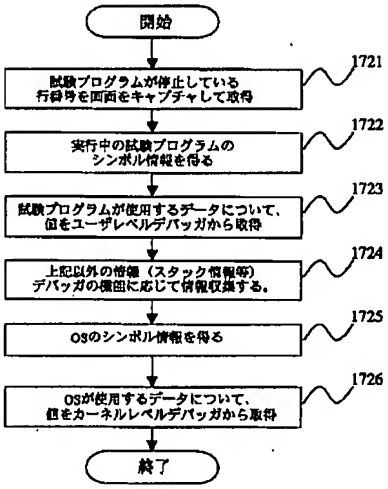
【図10】



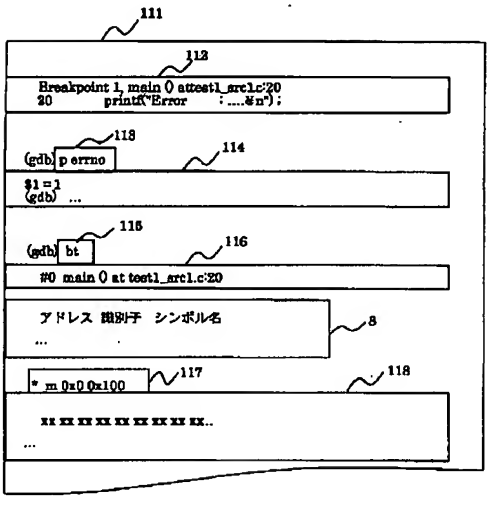
【図11】



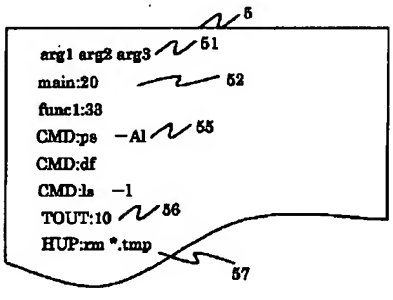
【図13】



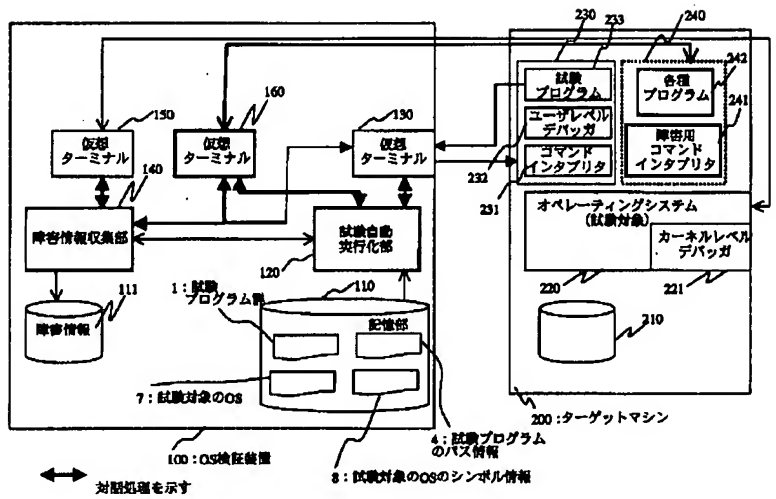
【図14】



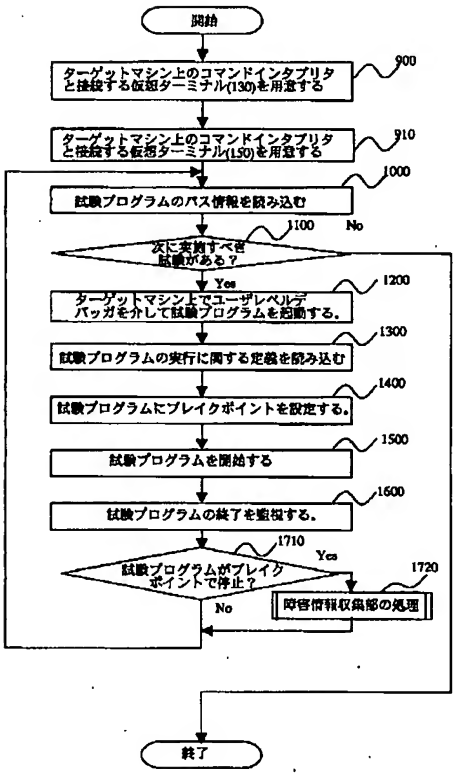
【図20】



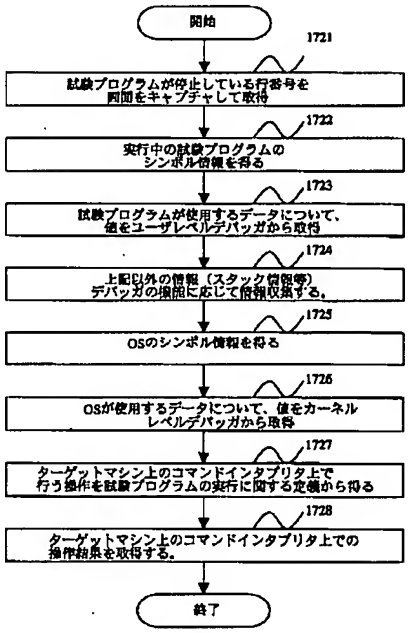
【図15】


230
233
240
242
241
試験プログラム
各種プログラム
ユーザレベルデバッガ
障害用コマンドインタプリタ
コマンドインタプリタ
232
231
オペレーティングシステム（試験対象）
カーネルレベルデバッガ
220
221
210
200：ターゲットマシン
150
160
190
仮想ターミナル
仮想ターミナル
仮想ターミナル
160
障害情報収集部
試験自動実行化部
120
111
障害情報
1：試験プログラム群
110
記憶部
7：試験対象のOS
100：OS検証装置
4：試験プログラムのパス情報
8：試験対象のOSのシンボル情報
対話処理を示す

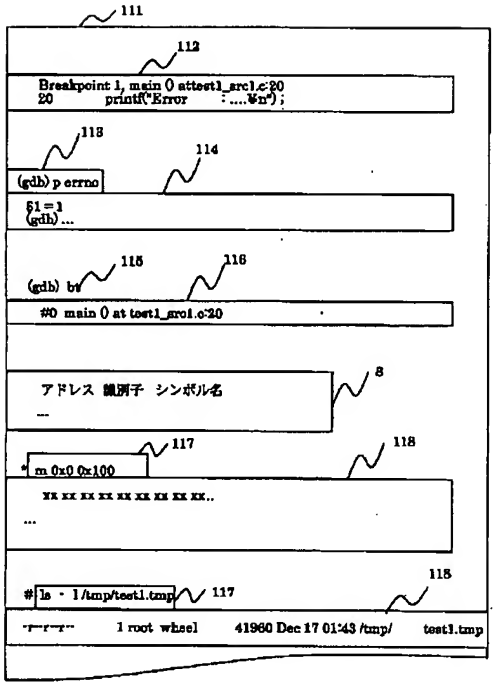
【図17】


開始
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(130)を用意する
900
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(150)を用意する
910
試験プログラムのパス情報を読み込む
1000
次に実施すべき試験がある？
1100
No
Yes
ターゲットマシン上でユーザレベルデバッガを介して試験プログラムを起動する。
1200
試験プログラムの実行に関する定義を読み込む
1300
試験プログラムにブレイクポイントを設定する。
1400
試験プログラムを開始する
1500
試験プログラムの終了を監視する。
1600
試験プログラムがブレイクポイントで停止？
1710
Yes
No
障害情報収集部の処理
1720
終了

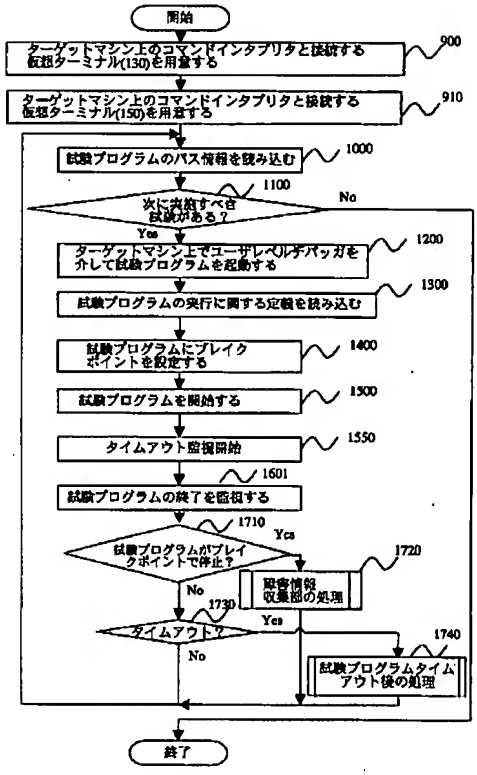
【図18】


開始
試験プログラムが停止している行番号を両面をキャプチャして取得
1721
実行中の試験プログラムのシンボル情報を得る
1722
試験プログラムが使用するデータについて、値をユーザレベルデバッガから取得
1723
上記以外の情報（スタック情報等）デバッガの機能に応じて情報収集する。
1724
OSのシンボル情報を得る
1725
OSが使用するデータについて、値をカーネルレベルデバッガから取得
1726
ターゲットマシン上のコマンドインタプリタ上で行う操作を試験プログラムの実行に関する定義から得る
1727
ターゲットマシン上のコマンドインタプリタ上での操作結果を取得する。
1728
終了

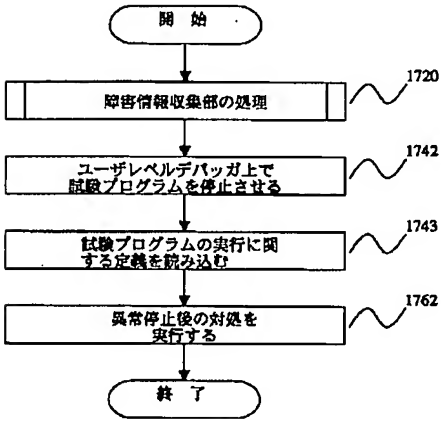
【図19】



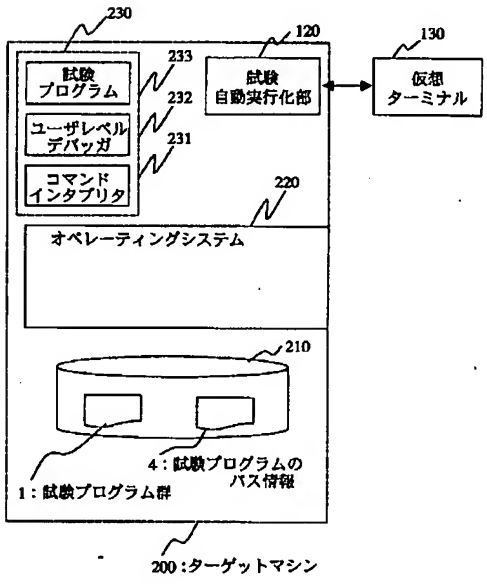
【図21】



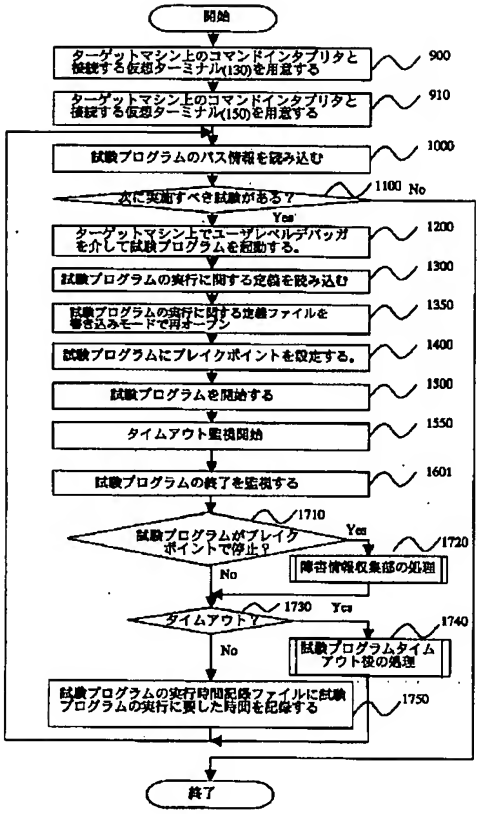
【図26】



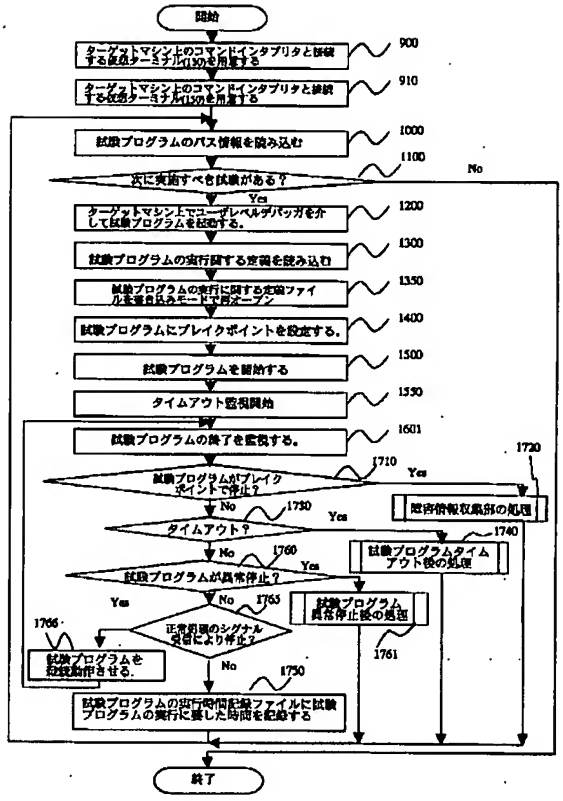
【図30】



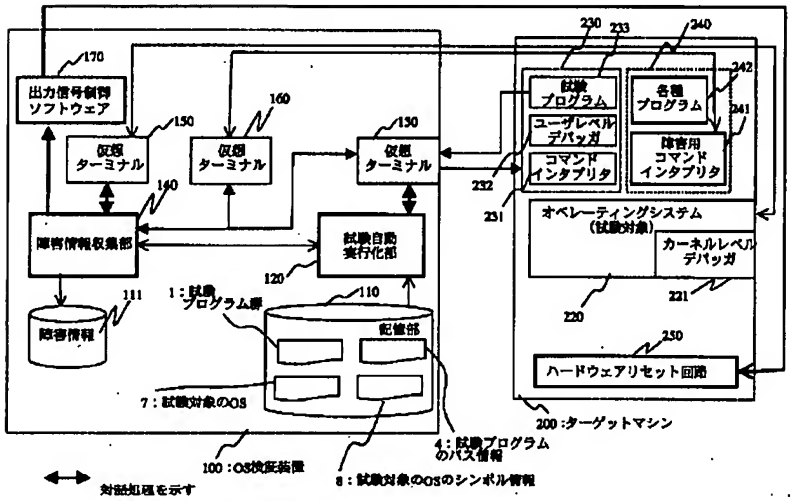
【図23】


開始
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(130)を用意する
900
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(150)を用意する
910
試験プログラムのパス情報を読み込む
1000
次に実施すべき試験がある？
1100
No
Yes
ターゲットマシン上でユーザレベルデバッガを介して試験プログラムを起動する。
1200
試験プログラムの実行に関する定義を読み込む
1300
試験プログラムの実行に関する定義ファイルを書き込みモードで再オープン
1350
試験プログラムにブレイクポイントを設定する。
1400
試験プログラムを開始する
1500
タイムアウト監視開始
1550
試験プログラムの終了を監視する
1601
試験プログラムがブレイクポイントで停止？
1710
Yes
1720
障害情報収集部の処理
No
タイムアウト？
1730
Yes
1740
試験プログラムタイムアウト後の処理
No
試験プログラムの実行時間記録ファイルに試験プログラムの実行に要した時間を記録する
1750
終了

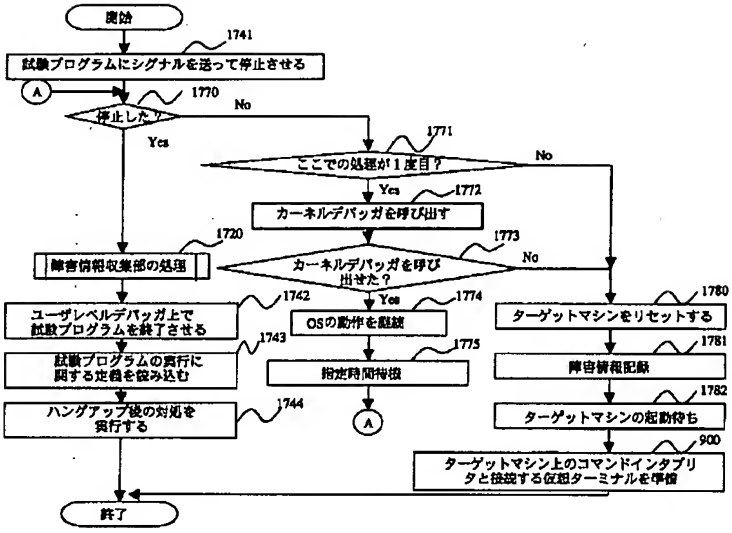
【図25】


開始
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(130)を用意する
900
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナル(150)を用意する
910
試験プログラムのパス情報を読み込む
1000
次に実施すべき試験がある？
1100
No
Yes
ターゲットマシン上でユーザレベルデバッガを介して試験プログラムを起動する。
1200
試験プログラムの実行に関する定義を読み込む
1300
試験プログラムの実行に関する定義ファイルを書き込みモードで再オープン
1350
試験プログラムにブレイクポイントを設定する。
1400
試験プログラムを開始する
1500
タイムアウト監視開始
1550
試験プログラムの終了を監視する。
1601
試験プログラムがブレイクポイントで停止？
1710
Yes
1720
障害情報収集部の処理
No
タイムアウト？
1730
Yes
1740
試験プログラムタイムアウト後の処理
No
試験プログラムが異常停止？
1760
Yes
No
試験プログラム異常停止後の処理
1761
正常範囲のシグナル受信により停止？
1765
No
試験プログラムを初期動作させる。
1766
試験プログラムの実行時間記録ファイルに試験プログラムの実行に要した時間を記録する
1750
終了

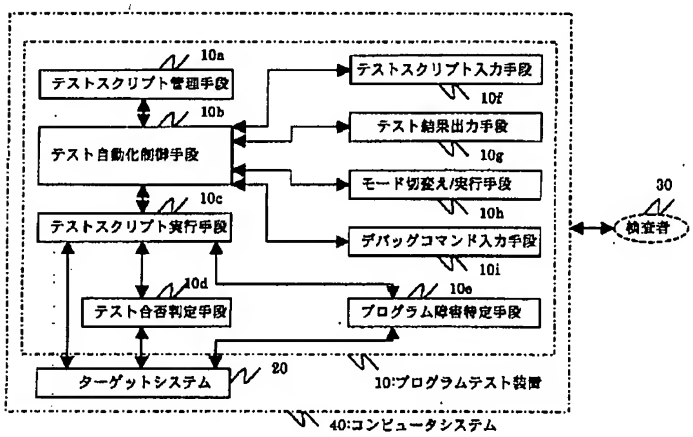
【図27】


出力信号制御ソフトウェア
170
仮想ターミナル
150
仮想ターミナル
160
仮想ターミナル
130
障害情報収集部
140
試験自動実行化部
120
障害情報
111
1：試験プログラム群
110
記憶部
7：試験対象のOS
100：OS検証装置
4：試験プログラムのパス情報
8：試験対象のOSのシンボル情報
対話処理を示す
230
233
240
試験プログラム
ユーザレベルデバッガ
コマンドインタプリタ
232
231
各種プログラム
242
241
障害用コマンドインタプリタ
オペレーティングシステム（試験対象）
カーネルレベルデバッガ
220
221
250
ハードウェアリセット回路
200：ターゲットマシン

【図29】


開始
1741
試験プログラムにシグナルを送って停止させる
A
1770
停止した？
No
Yes
1771
ここでの処理が1度目？
No
Yes
1772
カーネルデバッガを呼び出す
1773
カーネルデバッガを呼び出せた？
No
Yes
1720
障害情報収集部の処理
1742
ユーザレベルデバッガ上で試験プログラムを終了させる
1743
試験プログラムの実行に関する定義を読み込む
1744
ハングアップ後の対処を実行する
1774
OSの動作を継続
1775
指定時間待機
A
1780
ターゲットマシンをリセットする
1781
障害情報記録
1782
ターゲットマシンの起動待ち
900
ターゲットマシン上のコマンドインタプリタと接続する仮想ターミナルを準備
終了

【図31】


10a
テストスクリプト管理手段
10b
テスト自動化制御手段
10c
テストスクリプト実行手段
10d
テスト合否判定手段
ターゲットシステム
20
テストスクリプト入力手段
10f
テスト結果出力手段
10g
モード切変え/実行手段
10h
デバッグコマンド入力手段
10i
10e
プログラム障害特定手段
10:プログラムテスト装置
40:コンピュータシステム
30
検査者